Departamento de Informática

TRABAJO FIN DE GRADO

Prototipo de evaluación de una red de sensores inalámbrica basada en el protocolo 6LoWPAN

Autor: Alejandro José Tébar Martín

Tutor: María Soledad Escolar Díaz

Leganés, Septiembre de 2013

Índice

Índice de tablas

Índice de ilustraciones

Abstract

This Ending Degree Project introduces 6lowpan as a communication protocol in a wireless sensor network.

A Wireless Sensor Network (WSN) is a set of autonomous devices called sensor nodes or motes capable of using sensors for taking measurements of the area where they are deployed. The deployment, in addition, is conditioned by the radio range of the devices. WSNs are also limited by physical restrictions, such as the available memory, battery, as well as low computing capacity and a low data transmission rate.

With the aim of evaluating the correct operation of the 6lowpan protocol in a WSN, a prototype of wireless sensor network will be developed. The prototype will be integrated by three main elements: sensor nodes, *gateway*, and base station. *Sensor nodes* take readings and send these data to the base station. The *gateway* acts as a bridge between the sensor network and the base station, and enables communication between the two parts. Finally, a base station that will be the final device where sensor node data will be received and stored in a data base.

Using 6LoWPAN protocol motes will send two types of messages. The first one, contains the temperature measurement obtained using a sensor board. The second one collects statistics at different network stack layers.

This prototype will be validated through several scenarios that will test the different features offered by the prototype implemented. The result of the evaluation shows the correct functioning of the prototype as well as the advantages and disadvantages of the use of using 6LoWPAN.

Resumen

Este trabajo de fin de grado introduce 6LoWPAN como protocolo de comunicación dentro de una red de sensores inalámbrica.

Una red de sensores inalámbrica o Wireless Sensor Network (WSN) es un conjunto de dispositivos autónomos capaces de emplear sensores para tomar mediciones en la zona donde son desplegados. El despliegue, además, está condicionado por el radio de acción de los dispositivos. Las redes de sensores inalámbricas están limitadas también por las restricciones físicas, como la memoria disponible o la duración de las baterías, así como por una baja capacidad de cómputo y tasa de datos.

Para validar el correcto funcionamiento del protocolo 6LoWPAN se empleará un prototipo de red de sensores inalámbrica. El prototipo estará formado por tres elementos. Primero, nodos sensores que tomen lecturas y envíen los datos a la estación base. Segundo, un nodo *gateway* que actúe como puente entre la red de sensores y la estación base y permita que se produzca la comunicación entre las dos partes. Por último, una estación base que será el dispositivo final donde se recibirán los datos que envían los nodos sensores y se almacenaran en una base de datos.

Utilizando el protocolo 6LoWPAN los nodos sensores enviarán dos tipos de mensaje. El primero, contendrá la medición de temperatura que haya obtenido mediante una placa de sensores. El segundo, recopilará estadísticas sobre los protocolos de red.

Esto prototipo será validado a través de una serie de escenarios que prueban las diferentes funcionalidades que ofrece la implementación desarrollada. Los resultados de la evaluación muestran que el funcionamiento del prototipo desarrollado es correcto así como las ventajas e inconvenientes del uso de 6LoWPAN.

1. Introducción

A continuación se presenta el capítulo de introducción, donde se verá la motivación que nos han llevado a realizar este trabajo así como los objetivos que nos hemos marcado. Además, se incluirá un breve resumen de los diferentes capítulos que forman este trabajo y un glosario con los términos y conceptos más importantes, para posible consulta.

1.1 Alcance del proyecto

Una red de sensores inalámbrica, más conocida por su acrónimo inglés WSN (Wireless Sensor Network), es un conjunto de nodos que se emplazan en una zona concreta cuyo objetivo es la medición de diferentes variables a través de sensores específicos.

Las redes de sensores están compuestas por una cantidad variable de nodos sensores que se encargan de recopilar datos para enviarlos a una estación base que los recibe. Después, la estación base se encarga de almacenar o reenviar los datos, dependiendo de los requisitos del sistema.

Las redes de sensores son una tecnología en auge debido a que campos como la domótica están creciendo de forma exponencial. Además, como ejemplo de avance, en medicina se empiezan a ver redes encargadas de controlar el estado de un grupo de pacientes, permitiendo que el doctor pueda conocer de primera mano el estado en el que están.

En la actualidad, la parte de comunicación de la mayoría de redes de sensores usa el estándar de comunicación 802.15.4, que establece solo las dos primeras capas de la pila de protocolos OSI. Así, los nodos dentro de la red están sujetos a algunas limitaciones, como estar orientado a redes ad hoc de pequeño tamaño o estar asociado a un único estándar de radio.

Consideremos el siguiente ejemplo. Tenemos una red de sensores implementada mediante el estándar de comunicación 802.15.4. Esta red tendría el objetivo de controlar la temperatura en una zona industrial para evitar que pasara un umbral determinado. Al estar diseñada para usar el estándar de comunicación 802.15.4 tendríamos una serie de limitaciones, como es la cantidad de nodos sensores que puede tener la red en comparación con la enorme cantidad que ofrecería una implementación de IPv6. Esto, en caso de una zona amplia, sería un gran problema y probablemente nos obligaría a rediseñar la red.

En cambio si se implementara otro protocolo se obtendrían una serie de ventajas muy interesantes. Para la realización de este trabajo se ha elegido 6LoWPAN (*IPv6 over Low Power Wireless Personal Area Network*), que es un estándar que permite emplear IPv6 en redes de área personal con tasas bajas de transmisión de datos, también conocidas bajo el acrónimo en inglés LR-WAN. Es un estándar basado en el estándar de comunicación 802.15.4, un estándar de comunicación que define el nivel físico y el control de acceso al medio para LR-WAN.

La implementación de 6LoWPAN tiene una serie de ventajas que lo hacen muy atractivo. Primero, emplea la infraestructura y la tecnología IP, por tanto permite que las WSN dispongan de una cantidad de direcciones mucho mayor lo cual permite que existan más dispositivos. Además, permite emplear las topologías de malla y estrella, lo cual permite diferentes tipos de enrutamiento. También implementa mecanismos de manejo de direcciones, lo cual permite la comunicación entre diferentes estándares, como puede ser el 802.15.4 e IPv6. Para finalizar, mencionar que incluye algunas mejoras en seguridad.

En una red de sensores que esté implementada mediante el estándar de comunicación 802.15.4 los nodos de la red emiten la información al medio, mediante el canal de radio, y son los propios nodos los encargados de tener que escuchar para obtenerla.

Las limitaciones que trae consigo 6LoWPAN comienzan con el tamaño de las direcciones que maneja IPv6, las cuales pueden llegar a ser casi diez veces mayores que las empleadas por una WSN que use el estándar de comunicación 802.15.4. También es necesario incluir un servicio de descubrimiento de vecinos al emplear 6LoWPAN.

Lo que se pretende conseguir con este proyecto es la creación de un prototipo de WSN que sea capaz de emplear el estándar 6LoWPAN para la comunicación. Además, el prototipo será capaz de hacer que un nodo concreto sea capaz de comunicarse con otro nodo de la red.

1.2 Objetivos del proyecto

La finalidad de este proyecto es el diseño e implementación de un prototipo de red de sensores inalámbricos que incluya soporte IPv6 mediante 6LoWPAN.Para ello se desarrollará una aplicación en el sistema operativo para nodos sensores usando el sistema operativo TinyOS, cuya implementación tiene soporte para 6LoWPAN, y se instalará en los nodos sensores. Esta aplicación deberá ser capaz de enviar datos utilizando la tecnología IPv6. Por último, se validará el prototipo mediante una batería de pruebas que acoten las funcionalidades de la red de sensores.

Las fases por las que pasará el prototipo son las siguientes:

- Investigación de la tecnología: en esta fase se hace una comparativa entre la tecnología existente y que se puede aplicar al desarrollo del prototipo, con el fin de seleccionar la más adecuada.
- Toma de requisitos: se acotarán las acciones que debe realizar el prototipo.
- Desarrollo e implementación de la aplicación: en esta sección se diseñará la aplicación y se construirán las diferentes partes que la componen.
- Evaluación del prototipo: se utilizará una batería de pruebas para comprobar que el prototipo funciona correctamente.

Una vez vistas las fases, pasamos a ver el objetivo principal y los objetivos secundarios:

- Uso de 6LoWPAN como protocolo de comunicación: este es el objetivo primordial, ya que es el elemento que marca la diferencia con las redes de sensores actuales.
- Diseño e implementación de una aplicación que permita enviar datos de los nodos sensores a una estación base utilizando 6LoWPAN: será necesario desarrollar e implementar una aplicación que se instale en los nodos sensores para permitir la comunicación dentro de la red.
- Almacenamiento de los datos en la estación base: la estación base necesitará de un mecanismo que sea capaz de guardar los datos que recibe desde los nodos sensores de la red.
- Despliegue de la aplicación en una red de sensores física: el prototipo será evaluado en una red de sensores física ya que es la mejor manera de realizar su validación.

1.3 Estructura del documento

El documento está formado por siete capítulos donde se describe este trabajo. Después, se encuentran los anexos, donde se muestran temas relacionados con el trabajo pero que no forma una parte de él, como puede ser la instalación del entorno donde se han realizado las pruebas.

En este primer capítulo, *Introducción*, se da una visión general del trabajo desarrollado. Se proporciona una breve explicación de los capítulos que vendrán a continuación. Se incluye un apartado con la terminología y los conceptos básicos para mejor entendimiento de este trabajo.

En el capítulo dos, el *Estado de la Cuestión*, se describe la tecnología de las redes de sensores junto con los lenguajes de programación que se utilizarán, así como los protocolos de comunicación que se utilizarán.

El capítulo tres, *Especificación de requisitos*, está dedicado a los requisitos que deberá cumplir este trabajo. La sección está divida en dos partes: los que especifican las características de la red de sensores y los que delimitan las características que harán posible que la red de sensores funcione correctamente.

En el capítulo cuatro, *Decisiones de Diseño*, se detallarán los motivos de la elección de los diferentes elementos que forman este trabajo, como los elementos físicos que forman la red o los lenguajes que se utilizarán para la programación de la misma.

En el capítulo cinco, *Construcción*, se hablará de la fase de implementación del prototipo. Aso, se podrá comprender mejor el diseño desarrollado y cómo funcionan las partes más importantes del prototipo.

En el capítulo seis, *Evaluación*, se detallará la batería de pruebas y los resultados obtenidos con ellas, para poder obtener los datos de cara a las conclusiones.

En el capítulo siete, *Conclusiones*, explicaremos las conclusiones obtenidas con este trabajo.

Por último, se incluirán cuatro anexos como material complementario. En el Anexo A se detallará la instalación del sistema operativo Ubuntu y de TinyOS. En el Anexo B se incluirá el código de los scripts que forman la base de datos. En el Anexo C se incluye el código en el lenguaje Python encargado de recibir los datos e insertarlos en la base de datos. Para finalizar, el Anexo D contiene el código que se ha instalado en los nodos sensores que forman la red.

1.4 Terminología y conceptos básicos

6LoWPAN

IPv6 over Wireless Personal Area Network o IPv6 sobre Redes de Área Personal Inalámbricas, es un protocolo que posibilita el empleo de IPv6 en redes que emplean el estándar de comunicación IEEE 802.15.4.

802.15.4

El estándar IEEE 802.15.4 se encarga de definir el nivel físico y el control de acceso al medio para LR-WAN, redes inalámbricas de área personal con bajas tasas de transferencia de datos.

Estación base

Es el dispositivo final que participa en la red de sensores inalámbrica. Suele ser un PC y se encarga de la recepción y almacenamiento de datos.

ICMP

Internet Control Message Protocol o Protocolo de Mensajes de Control de Internet es el sub protocolo de control y notificación de errores del Protocolo de Internet.

IP

Internet Protocol o Protocolo de Internet es un protocolo de comunicación de la capa de red. Es un estándar que se emplea para enviar y recibir información mediante una red que emplea paquetes conmutaos.

IPv6

Internet Protocol version 6 o versión 6 del Protocolo de Internet, es la última versión del protocolo de comunicación IP que provee de un identificador y un sistema de localización a los computadores pertenecientes a redes conectadas a Internet.

MicaZ

Es un modelo de nodo sensor.

nesC

Es una variante del lenguaje de programación C dirigido a la programación de sensores, donde la limitación de la memoria hace que el propio lenguaje no sea una opción a considerar.

Nodo gateway

Este nodo se encarga de hacer de nexo entre la red de sensores y la estación base.

Nodo sensor

Un nodo sensor es un dispositivo perteneciente a una red de sensores inalámbrica que es capaz de procesar y recabar información dentro del entorno así como de comunicarse con otros nodos de la red.

PAN

Personal Area Network o Red de Área Personal, son redes de dispositivos para con un radio de acción limitado a pocos metros.

Placa de sensores

Una placa de sensores es un dispositivo auxiliar que se acopla a un nodo sensor para proveerle de una serie de sensores. El modelo empleado en este trabajo, MTS300, provee de sensores para medir temperatura, luminosidad y sonido.

TCP

Transport Control Protocol o Protocolo de Control de Transporte, es un protocolo de nivel de transporte orientado a conexión. Además, proporciona mecanismos para garantizar la entrega de los datos en orden y sin errores.

TCP/IP

Transmission Control Protocol / Internet Protocol, es el estándar de comunicaciones en red, empleado para la interconexión de sistemas informáticos a través de Internet.

TinyOS

Es un sistema operativo de código abierto diseñado para redes de sensores inalámbricas. Está diseñado para funcionar bajo las restricciones que impone el trabajo con sensores, como es la baja cantidad de memoria disponible. Emplea el lenguaje nesC para la implementación de aplicaciones.

UDP

User Datagram Protocol o Protocolo de Datagramas de Usuario, es un protocolo del nivel de transporte que permite el envío de datagramas sin establecer una conexión previa, como sí

hace TCP. Tampoco tiene control de flujo, por tanto su empleo se centra en entornos donde la pérdida de paquetes no sea un problema.

WSN

Wireless Sensor Network o Red de Sensores Inalámbrica, es un conjunto de dispositivos que se emplazan en una zona concreta cuyo objetivo es la medición de diferentes variables a través de sensores específicos.

Zigbee

Es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica, basada en el estándar 802.15.4 de redes inalámbricas de área personal o WPAN. Se centra en aplicaciones que tengan una baja tasa de envío de datos lo que permite que tengan una autonomía alta.

2. Estado de la cuestión

En este capítulo se dará una visión general del estado actual de las diferentes tecnologías implicadas en el ámbito de las redes de sensores. Se describirán distintas opciones y en el capítulo cuatro se explicarán las decisiones de diseño y los motivos por los que se ha seleccionado el elemento que se empleará.

2.1 Redes de sensores

Las WSN (*Wireless Sensor Network*), o Redes de Sensores Inalámbricos, son redes ad-hoc compuestas por dispositivos sensoriales autónomos que están distribuidos regular o aleatoriamente dentro de un área determinada para monitorizar diferentes condiciones, como puede ser la temperatura o la humedad. A estos dispositivos se les da el nombre de nodos, aunque en inglés también se les llama "motes" en algunas ocasiones.

El origen de las redes de sensores es similar al de internet, ya que está relacionado con el ámbito de defensa y con DARPA. Comenzó con DSN (*Distributed Sensor Networks*) [1] alrededor de 1980. La idea principal era la de tener una red de nodos de bajo coste conectados entre sí pero que funcionaran de manera autónoma para que pudiera transmitir la información al nodo que mejor pudiera aprovecharla.

A partir del año 2000 se produce un gran incremento en la investigación de esta área por la increíble cantidad de aplicaciones potenciales que existen. Debido a esto comienza la comercialización de sensores.

Los sensores tienen una gran cantidad de características que les permiten diferenciarse entre ellos, como pueden ser el tamaño o la capacidad de almacenamiento. De este modo, se pueden emplear diferentes modelos para diferentes tareas y así poder obtener un mejor rendimiento.

2.1.1 Componentes de una WSN

Una red de sensores inalámbrica cuenta con tres elementos bien diferenciados: nodos sensores, nodo *gateway* y estación base. En la Ilustración 1 se puede ver la idea general de la arquitectura de la red y a continuación se describe cada uno de los elementos.

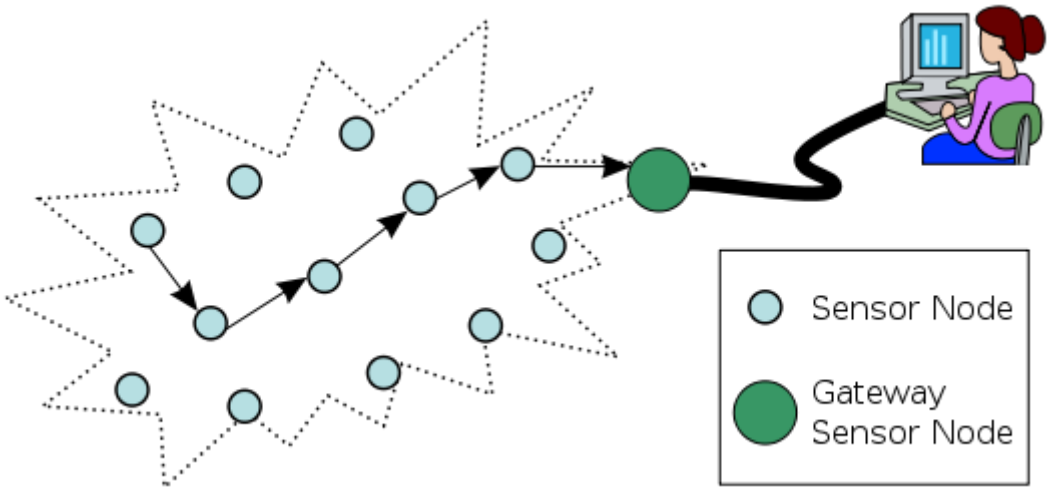

Ilustración 1: Esquema WSN

A continuación se explicará el funcionamiento general de los componentes que forman la red. En el capítulo cuatro, se explicarán las decisiones de diseño de los componentes físicos, por tanto será ahí donde se den las características más técnicas.

2.1.1.1 Nodo Sensor

Un nodo sensor es un dispositivo perteneciente a una red de sensores inalámbrica que es capaz de procesar y recabar información dentro del entorno así como de comunicarse con otros nodos de la red. Su estructura hardware está compuesto por los siguientes elementos:

- **Micro controlador:** suele tener un solo núcleo debido a las restricciones que se aplican al trabajo con sensores.
- **Radio:** permite la comunicación, transmisión y recepción de datos, con los demás elementos de la red.
- **Sistema de memoria:** suele estar compuesto por tres elementos: memoria flash, memoria RAM y memoria EEPROM. La memoria flash se emplea para almacenar los datos obtenidos en las mediciones y es una memoria externa al micro controlador. En la memoria EEPROM se guardan los datos referentes a la programación del nodo. La memoria RAM suele ser muy reducida y hay modelos que no disponen de ella. La memoria EEPROM y RAM se localizan dentro del micro controlador.
- **Fuente de energía:** puede ser una batería, pilas o algún elemento que le permita recolectar la energía como puede ser una placa solar.
- **Sensores:** disponen de uno o varios sensores para realizar mediciones ambientales, como humedad o temperatura.

Si hablamos de modelos de sensores podemos encontrar una gran variedad de ellos. Esto se debe a que, al existir numerosos motivos para usar nodos, van surgiendo nuevos modelos que puedan satisfacer los huecos de mercado. Por ejemplo, no todos los nodos dispondrán de la misma memoria ni de la misma fuente de alimentación. Solo con una de estas dos características se podrían realizar numerosos sensores diferentes.

Además, hay que considerar que algunos modelos de nodo sensor no traen de inicio sensores, como los modelos MicaZ [2]. Para solucionar este problema existen placas de sensores que pueden acoplarse al nodo mediante un conector de expansión de 51 pines. Así, se puede añadir una funcionalidad adicional a ese nodo.

En la Ilustración 2 se puede ver un nodo sensor MicaZ.

Ilustración 2: Nodo sensor MicaZ

Existen una serie de limitaciones a la hora de trabajar con los nodos sensores. Primero, debido a su tamaño acarrean una serie de limitaciones hardware, como puede ser la cantidad de datos que puede almacenar un nodo sensor debido a su reducida memoria flash. Esto también se puede relacionar con la limitación software que se produce debido a que las aplicaciones que se instalan en los nodos sensores no pueden ser complejas debido a las limitaciones de memoria y del propio micro procesador.

Otro ámbito donde surge la limitación es el tiempo que puede permanecer en activo. Si debe incluirse una batería la duración es limitada y en el momento en que la batería se agote habría que cambiarla manualmente, lo cual puede ser un problema.

2.1.1.2 Nodo Gateway

Este nodo se encarga de hacer de nexo entre la red de sensores y la estación base. Este tipo de nodo tiene algunas características diferentes, ya que por ejemplo, no suele disponer de sensores.

Principalmente se encargan de recibir los datos de la red de sensores para reenviarlos a su destino, normalmente una estación base. En algunos casos el nodo *gateway* está unido a un nodo sensor, el cual no dispone de sensores pero le permite comunicarse con la WSN.

A continuación, en la Ilustración 3 se puede ver un ejemplo de nodo *gateway*.

Ilustración 3: Nodo Gateway MIB520

2.1.1.3 Estación base

Por último, hablaremos de la estación base. Es el dispositivo final que participa en la red de sensores inalámbrica. Puede ser un PC o un elemento que tenga el hardware y el software necesario para poder enviar y recibir los menajes que emplea la WSN. Además, se encargará de almacenar los datos, ya sea en un fichero de texto o en una base de datos.

En algunos casos, este elemento sólo se encarga de funcionar como otro nodo más y procesar la información para ser reenviada al destino deseado.

2.1.2 Características de las WSN

Las principales características de las WSN son:

- **Topología dinámica:** una red de sensores dispone de una topología que cambia de forma constante en el tiempo, y por tanto los nodos deben ser capaces de adaptarse.
- **Variabilidad del canal:** el canal de comunicación, radio, es variable debido a los diferentes fenómenos que le pueden afectar, como son: atenuación de la señal; interferencias o desvanecimientos, ya sean rápidos (causados por estar los elementos que se comunican demasiado cerca entre ellos) o lentos (existe algún obstáculo que impide que la transmisión sea clara).
- **No se usa infraestructura de red:** los nodos sensores pueden funcionar como emisor, receptor o incluso como enrutador. Todo esto es posible gracias a que existe un nodo especial, conocido como nodo recolector, que se encarga de recopilar toda esta información para poder enviarla al dispositivo final, que suele ser un PC conectado a ese nodo.
- **Tolerancia a errores:** un nodo debe mantenerse funcionando incluso cuando existen problemas en su sistema. Además, no puede permitirse que un error en un nodo haga que la red deje de funcionar correctamente.
- **Comunicaciones multisalto o *broadcast*:** una red de sensores admite tanto los protocolos basados en el multisalto (por ejemplo: AODV[3] o DSDV[4]) como los basados en el *broadcast*.

- **Consumo energético:** en las redes de sensores inalámbricos es necesario que los sensores estén dispersos, por tanto no es posible que estén conectados a una gran fuente de energía. Para que estén en funcionamiento se utilizan fuentes de energía portátiles, como una batería o una pila energética. También, puede disponer de elementos que le permitan cosechar su propia energía, como pueda ser una pequeña placa solar fotovoltaica.
- **Limitaciones de hardware:** es necesario que el hardware esté limitado para poder crear elementos independientes con una alta autonomía, lo cual se consigue limitando elementos como la memoria. Así, el consumo energético será mínimo y podrán mantenerse en funcionamiento de manera independiente mucho más tiempo.
- **Costes de producción:** el coste de producción es bajo debido a las limitaciones de software, gracias a lo cual se pueden generar grandes redes de sensores a un bajo coste.
- **Heterogeneidad de los nodos:** existe una gran cantidad de sensores diferentes, en función de las características que sean necesarias para la tarea que se vaya a llevar a cabo.
- **Capacidad para soportar duras condiciones ambientales:** si los sensores no van a estar funcionando en un entorno controlado, como pueda ser al aire libre, debe ser resistente a las condiciones atmosféricas de la zona, para evitar que vida efectiva sea corta y pueda desempeñar su trabajo.

2.1.3 Aplicaciones de las WSN

Algunas de las áreas donde este tipo de redes tiene una gran acogida son las siguientes:

- **Monitorización de una zona:** la red se despliega en una zona donde monitorizará un fenómeno concreto, como puede ser el transito civil en un área delimitada. Además, se puede orientar el funcionamiento de la red para obtener los datos con los que predecir movimientos, como por ejemplo los gases. También, se puede llevar a entornos naturales para controlar que especies vegetales, como los viñedos, crezcan de forma controlada y con un coste menor, ya que se puede controlar el gasto del riego entre otras cosas.
- **Detección de condiciones ambientales:** las redes de sensores pueden colocarse al aire libre para realizar tareas de medición o de monitorización:
    - Calidad medioambiental: puede destinarse al control de los niveles de ciertos elementos nocivos en el aire o el agua para controlar su calidad.
    - Prevención de desastres naturales: como las inundaciones o los incendios.
    - Monitorización de elementos: como pueden ser especies animales o elementos naturales como los glaciares.

En la Ilustración 4 se puede ver cómos se ha colocado una red de sensores para monitorizar la temperatura de una zona.

Ilustración 4: WSN que controla la temperatura de una zona

- **Domótica:** con las casas inteligentes hay numerosos factores a considerar en la propia casa:
  - Energía: se puede incluir de numerosas formas para mejorar el consumo, como por ejemplo, destinando la mayor cantidad posible de tareas que requieran suministro continuado de energía a horarios con tarifa reducida.
  - Facilidades: gestionando las luces de la casa en función de la luz ambiental o automatizándolo para que solo se mantengan encendidas cuando haya gente presente son algunos ejemplos.
  - Seguridad: esta parte se puede relacionar con el punto anterior si lo orientamos a sensores que puedan evitar que se produzcan incendios o detectar las fugas de gas. También puede tomar el camino de la detección de personas para evitar que se produzcan asaltos a la vivienda mediante el cerrado automático de puertas.
- **Medicina:** el uso de las redes de sensores en medicina tiene muchas posibles aplicaciones, tales como:
  - Monitorización de pacientes: para poder controlar sus constantes y poder modificar el tratamiento que se les administra para hacerlo más efectivo.
  - Monitorización de los niveles de determinadas sustancias en la sangre: puede aplicarse directamente a enfermedades como la diabetes para ver el nivel de glucosa en sangre.
  - Tratamiento de algunas deficiencias en algunos órganos: como por ejemplo aplicar a la retina, sólo en caso de haber perdido el funcionamiento de ésta, para poder incrementar la visión del paciente.

En la Ilustración 5 se puede ver el esquema de una WSN enfocada a que un doctor pueda conocer en todo momento el estado de los pacientes.

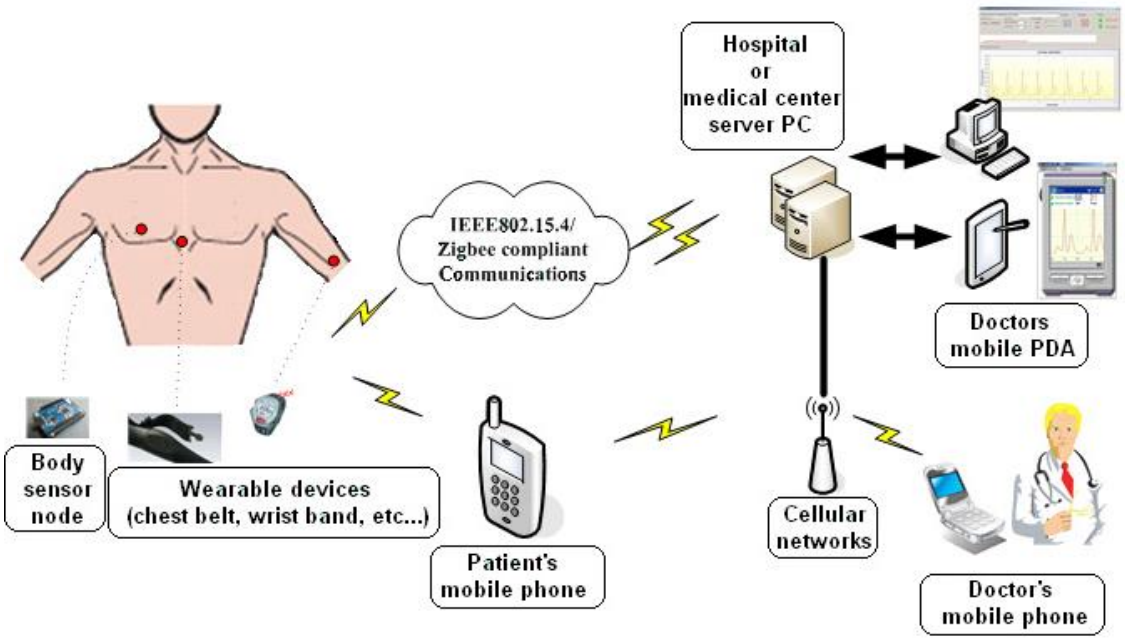

Ilustración 5: WSN con un enfoque médico

2.2 Entorno de Trabajo

En este apartado se describirán las herramientas de software que hemos utilizado para establecer el entorno de trabajo necesario para el desarrollo de nuestro proyecto.

2.2.1 TinyOS

TinyOS [5] es un sistema operativo de código abierto diseñado específicamente teniendo en cuenta las restricciones de los nodos sensores. Para ello utiliza el lenguaje nesC descrito más abajo en este documento. Al estar destinado a las redes de sensores debe adecuarse a las restricciones de memoria y otras limitaciones que existen en este ámbito.

Comenzó como una colaboración entre la universidad de California Berkeley, Intel Research y Crossbow Technology. Ahora es un consorcio conocido como la Alianza TinyOS.

TinyOS es un sistema operativo basado en eventos. Dispone de dos niveles de planificación: eventos y tareas.

Cuando se necesita que se ejecute un servicio, como puede ser el inicio de la función de lectura del sensor, se emplea un comando, el cual es similar a una función en C. El evento es la respuesta proporcionada por el hardware o el propio sistema operativo que indica que esa función ha concluido. Para cada evento, el programador generalmente implementa un manejador de evento que se ejecutará como respuesta al evento. Estos manejadores no tienen mucha carga de procesamiento y tienen la propiedad de que son capaces de interrumpir la ejecución de una tarea.

Las tareas poseen una carga mayor de procesamiento; este procesamiento no debe ser crítico en el tiempo debido a que si se interrumpe por un evento pueden producirse errores indeseados. El modelo de ejecución básico de las tareas es su ejecución hasta finalización en lugar de funcionar indefinidamente, lo cual hace que sean elementos más ligeros que los hilos.

Las tareas en este sistema operativo siguen una estructura FIFO (una cola) y obligatoriamente debe dar soporte a las tareas de nesC. La cola de tareas puede almacenar un máximo de 255 tareas.

Hasta el momento, hay dos versiones de este sistema operativo: 1.x y 2.x. La versión 2.x es la que incorpora las librerías para poder trabajar con 6lowpan. Para este trabajo se empleará la versión 2.1.1, que aunque no es la última sí nos permite emplear 6lowpan con los nodos MicaZ.

2.2.2 Mote Runner

Mote Runner [6] es un entorno de desarrollo para aplicaciones sobre WSN que además dispone de un simulador para sensores MEMSIC Iris [16].

El entorno se basa en la instalación de un programa adicional para Eclipse, que se encarga de permitir que los programas en los lenguajes Java y C# se puedan amoldar a las necesidades de los sensores.

Actualmente solo se puede instalar el entorno en Windows, aunque a largo plazo pretenden que se pueda hacer tanto en Linux como en sistemas Mac OS.

En cuanto al hardware de los sensores ocurre como con las plataformas, se da soporte a Iris pero esperan que en el futuro puedan darle soporte a otros grupos, como Telos o Mica.

2.2.3 Contiki

Contiki [7] es un sistema operativo de código abierto orientado a redes de dispositivos con baja carga de memoria. Debido a esto se ha diseñado para poder generar aplicaciones con necesidades de memoria muy bajas. Soporta nuevos estándares como 6LoWPAN, RPL o CoAP. También soporta IP de forma completa, permitiendo emplear TCP, UDP o HTTP.

Al contrario que Mote Runner, que solo da soporte a IRIS, Contiki dispone de soporte para una cantidad de hardware mayor, como por ejemplo MicaZ o WiSMote.

Incluye su propio simulador, Cooja [7], que le permite simular redes de nodos ejecutando aplicaciones desarrolladas con Contiki.

2.3 Lenguajes de Programación

2.3.1 nesC

nesC, (Network Embedded Systems C) [8], es una variante del lenguaje de programación C dirigido a la programación de sensores, donde la limitación de la memoria hace que el propio lenguaje no sea una opción a considerar.

Además, está diseñado para la programación de aplicaciones sobre el sistema operativo TinyOS. Los conceptos básicos que hay detrás de nesC son los siguientes:

- Los programas se construyen a partir de componentes que se ensamblan para conseguir un programa completo. Los componentes tienen dos partes: *configuración*, donde se incluyen los nombres de los interfaces, componentes y cómo se interrelacionan; e *implementación*, donde se incluye el código para su funcionamiento.
- Un componente puede proveer o usar interfaces. Las interfaces pretenden representar la funcionalidad del componente que las provee, mientras que las interfaces usadas representan la funcionalidad que el componente necesita para realizar su trabajo.
- Las interfaces son bidireccionales: especifican un conjunto de funciones a ser implementadas por el proveedor de las interfaces (comandos) y un conjunto a ser implementado por el usuario de la interfaz (manejador de eventos). Esto permite que una simple interfaz represente una interacción compleja entre componentes.
- Las tareas son funciones con una carga de procesamiento mayor que los eventos, que puede dejarse para que se procese en el futuro (asíncronas). Esto se realiza mediante la operación *post*, que inserta la tarea en la cola de tareas para que se ejecute más adelante.

El código en TinyOS es monoproceso, lo que quiere decir que si un elemento comienza su ejecución, no permitirá que otro elemento acceda a la CPU hasta que termine. La parte positiva de esto es que el gasto de memoria RAM es muy bajo. La parte negativa de esto es que, mientras un fragmento de código permanece ejecutándose en la CPU por un periodo de tiempo largo, comienzan a existir algunos problemas, como puede ser que el sensor tarde más tiempo en responder a un paquete.

nesC es un sistema no bloqueante, lo cual hace que sus operaciones de larga ejecución sean operaciones divididas en fase (*phase-split*). La operación dividida en fase toma este nombre al separar la invocación y la terminación en dos fases separadas de ejecución. En un sistema bloqueante, cuando un programa llama a una operación de larga ejecución, la llamada no retorna nada hasta que se ha completado, ya que la llamada bloquea el proceso. En el caso de un sistema dividido en fase, cuando se llama a una operación de larga ejecución la llamada vuelve inmediatamente y no bloquea el proceso. Es necesario un nuevo evento que indique que la operación ha terminado, obteniendo un retrollamada (*callback*). Por ejemplo, si queremos enviar algo invocamos el comando`send()` y posteriormente, el hardware del sistema generará el evento`sendDone()` para indicar que la operación ha concluido.

Para comprender mejor la diferencia entre el sistema bloqueante y el sistema no bloqueante o en fase dividida se ofrece a continuación un ejemplo en la Tabla 1:

Sistema bloqueante	Sistema en fase dividida

`If (send() == SUCCESS) {`
 	
`sendCount++;`
 	
`}`
	//fase de invocación
	
	`Send();`
	
	//fase de terminación
	
	`voidsendDone(error_terr) {`
	
	`If (err == SUCCESS) {`
	
	`sendCount++;`
	
	`}`
	
	`}`

Tabla 1: Sistema bloqueante y sistema no bloqueante

Los componentes están divididos en dos partes: *configuración* e i*mplementación*.

La *configuración* representa el ensamblado de las interfaces utilizadas y proporcionadas por un componente. La *implementación* es la parte donde se programarán las acciones que realizará nuestro componente. Está dividido a su vez en dos partes: *provides* son las interfaces que ofrece el componente e *implementation* es donde se programan las acciones que realiza el componente.

Utilizaremos un código muy sencillo, la aplicación Blink que viene por defecto al instalar TinyOS, para explicar mejor esta parte. Blink es una aplicación que utiliza tres temporizadores para encender tres *leds*.

Primero, el archivo de configuración, donde se muestran las diferentes interfaces que emplea.

```
configuration BlinkAppC
{
}
implementation
{
  components MainC, BlinkC, LedsC;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;


  BlinkC -> MainC.Boot;

  BlinkC.Timer0 -> Timer0;
  BlinkC.Timer1 -> Timer1;
  BlinkC.Timer2 -> Timer2;
  BlinkC.Leds -> LedsC;
}
```

Aquí se realizan tres acciones básicas. Primero, se establecen las interfaces que van a ser usadas por la aplicación, cuya parte corresponde a las líneas que comienzan por *components*. Después, algunas interfaces se renombran para poder trabajar con ellas, como ocurre con los temporizadores. Por último, las interfaces se cablean, en inglés *wiring*, para poder usarlas en nuestra aplicación. Un ejemplo de este *wiring* es `BlinkC.Timer0 -> Timer0`.

Ahora, se mostrará el archivo de implementación, que es el que contiene el código con las funcionalidades.

```
#include "Timer.h"

module BlinkC @safe(){
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Leds;
  uses interface Boot;
}
implementation{
  event void Boot.booted()  {
    call Timer0.startPeriodic( 250 );
    call Timer1.startPeriodic( 500 );
    call Timer2.startPeriodic( 1000 );
  }

  event void Timer0.fired()  {
    call Leds.led0Toggle();
  }

  event void Timer1.fired()  {
    call Leds.led1Toggle();
  }

  event void Timer2.fired()  {
     call Leds.led2Toggle();
  }
}
```

En este archivo de implementación se muestra un primer apartado, denominado *module*, donde se muestran las interfaces que van a ser usadas por los distintos componentes que usa la aplicación. Posteriormente, el apartado *implementation* contiene la implementación del componente, en particular de los manejadores de eventos, tareas y otras funciones. Una vez que el nodo sensor se ha puesto en funcionamiento con `Boot.booted()` se inician los tres temporizadores. Cada vez que uno de ellos termine, se accede a una de las tres funciones inferiores, en función de qué temporizador haya terminado. Esas funciones lo único que hacen es poner el *led* correspondiente a parpadear.

A continuación, se incluye la Ilustración 6 donde se puede apreciar un diagrama con las uniones de los diferentes elementos incluidos en Blink. Esto se realiza mediante cajas con diferentes bordes. En nesC las cajas simples son módulos, las dobles son configuraciones y las cajas que tienen los bordes discontinuos son componentes genéricos.

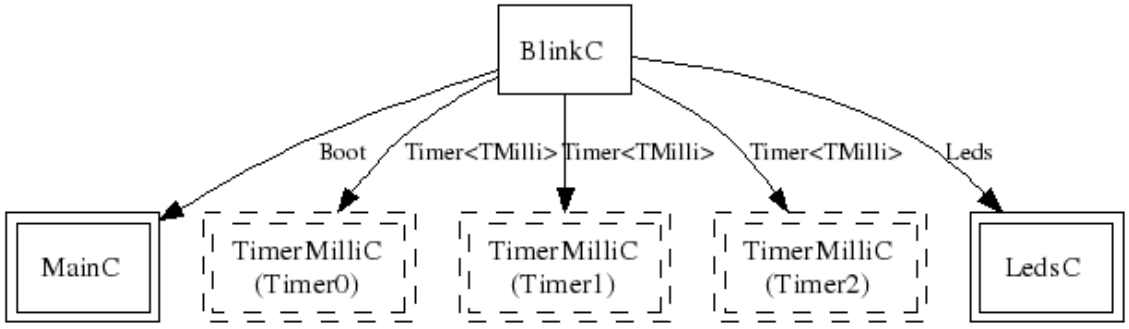

Ilustración 6: Diagrama de la aplicación Blink

2.3.2Python

Python [9] es un lenguaje de programación interpretado con muchos usos dentro del desarrollo de software. Es un lenguaje muy claro, con una sintaxis que favorece el código legible. Además, puede ser embebido en aplicaciones como una interfaz de script.

Python funciona sobre prácticamente cualquier sistema operativo bien conocido: Windows, Linux o Mac. Además, puede integrar diversos módulos externos, C o C++, para poder realizar tareas que no puede realizar el lenguaje en sí mismo. Esto también se extiende a las librerías, ya que por ejemplo se pueden instalar librerías Java.

Por último, Python es un lenguaje de código abierto, las licencias son emitidas por la fundación de software Python y son totalmente gratuitas.

2.4 Estándares de comunicación

2.4.1 IEEE 802.15.4

El estándar IEEE 802.15.4 [10] se encarga de definir el nivel físico y el control de acceso al medio para LR-WAN, redes inalámbricas de área personal con bajas tasas de transferencia de datos.

Este estándar sirve como base para otros estándares como son Zigbee, el cual se verá a continuación, o 6LoWPAN.

Este estándar está concebido para proporcionar comunicación a redes formadas por dispositivos con bajo coste y en una zona reducida. En cambios, es posible emplear estas redes en tiempo real, lo que abre una gran cantidad de posibilidades. También, dispone de un sistema CSMA/CA, un sistema de evasión de colisiones para evitar que se produzcan cogestiones en el tráfico de datos.

Los dispositivos que implementan el estándar de comunicación 802.15.4 pueden transmitir los datos que envían en tres frecuencias diferentes.

2.4.2 Zigbee

Zigbee [11] es un conjunto de protocolos orientado a las comunicaciones que tienen lugar en una WSN (red de sensores inalámbrica). Está basado en el estándar IEEE 802.15.4. Zigbee fue

desarrollado por la alianza Zigbee, un consorcio de empresas que se dedican principalmente a la industria de los semiconductores.

Emplea la radio frecuencia 2.4 GHz para enviar información a cualquier parte del mundo. Además, dispone de algunas frecuencias únicas en cada continente, como puede ser la banda 868 MHz en Europa.

Funciona en redes donde los sensores son de bajo coste y tienen un consumo bajo, lo que permite que estas redes tengan una autonomía alta y un tráfico de datos bajo.

Uno de los elementos con los que más se compara Zigbee es Bluetooth, debido a sus similitudes. Los principales puntos fuertes que presenta Zigbee es ofrecer redes con una cantidad mayor de dispositivos y un consumo energético menor. Por otro lado, Bluetooth dispone de una velocidad de transferencia mucho mayor.

2.5 IPv6

IPv6, *Internet Protocol version 6*, es la última versión de IP, *Internet Protocol*, el protocolo de comunicación que provee de un identificador y un sistema de localización a los computadores pertenecientes a redes conectadas a Internet.

2.5.1 Motivos de desarrollo

El predecesor de IPv6 es IPv4 [12], el cual apareció como un RFC, el número 791, en IETF en el año 1981. El espacio de direcciones disponible con IPv4 es de 4.294.967.296 (2^{32}) direcciones. Esta cantidad tenía los días contados debido al método que emplearon a la hora de repartir las direcciones. Así, se estimó que el espacio de direcciones estaría agotado a principios del año 2011, cosa que ocurrió.

Una vez que se supo que el espacio de direcciones de IPv4 iba a agotarse se comienza a pensar en posibles soluciones. IPv6 es una solución a largo plazo, que dispone de un espacio de direcciones de 128 bits, lo que permite tener 2^{128} direcciones, un espacio mucho mayor que es difícil que se agote a medio plazo.

En el año 2012 se habilitó de forma permanente IPv6 en los elementos de los principales proveedores de internet.

2.5.2 Asignación de direcciones

La IANA (Internet Assigned Numbers Authority) [13], se encarga de la repartición de las direcciones IPv6, como ya hizo con IPv4. Lleva repartiendo direcciones IPv6 dese el año 1995. Ahora mismo solo ésta disponible la octava parte de direccionamiento, ya que el resto se ha reservado para futuro.

El formato de la dirección IPv6 variará en función del tipo de dirección que sea manteniendo siempre los 128 bits. Por ejemplo, las direcciones de enlace-local y las *unicast* no tienen el mismo formato. Estos 128 bits se representan mediante ocho grupos de cuatro dígitos

hexadecimales, de esta forma cada uno de los ocho grupos representa 16 bits. Cada grupo se separa del siguiente mediante dos puntos (:), por ejemplo:

2001:0db8:5a3c:0000:0000:da6e:0990:74b5

Para poder interpretar una dirección IPv6 hay que tener en cuenta que pueden aparecer grandes cantidades de ceros, sobretodo formando grupos exclusivamente formados por ceros. Estos ceros iniciales de los grupos pueden ser eliminados, siempre y cuando se mantengan dígitos hexadecimales. Además, cuando existen grupos exclusivamente formados por ceros, como los grupos cuarto y quinto de la dirección de ejemplo superior, se pueden sustituir los grupos por un espacio en blanco seguido de dos (:). También, deben cumplirse las siguientes reglas:

- Solo pueden eliminarse grupos de ceros una vez. De este modo, no se producen direcciones ambiguas.
- Si se pueden eliminar varios grupos de ceros se eliminará la mayor cantidad de grupos posibles.
- Si el número de grupos a eliminar es el mismo, se modificarán los grupos más a la izquierda de la dirección.

Siguiendo todos estos pasos podemos modificar la dirección IPv6 de ejemplo como vemos en la Ilustración 7:

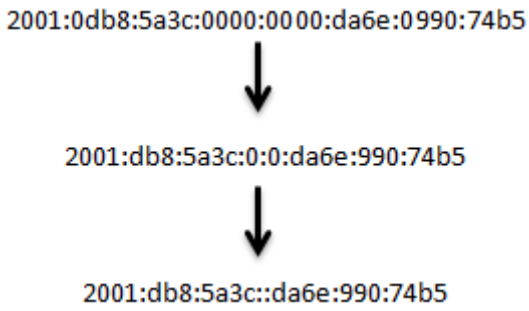

Ilustración 7: Simplificación dirección IPv6

En el primer paso se han eliminado los ceros iniciales de los grupos dejando siempre al menos un dígito hexadecimal. Después, se han eliminado los dos grupos de ceros dejando espacios vacíos dentro de dos (:).

También, mencionar que es posible representar direcciones IPv4-mapeadas mediante una notación muy similar a esta. Así, la dirección 192.0.2.128 se representaría en IPv6 mediante ::ffff:c000:280 o bien mediante ::ffff:192.0.2.128.

Una vez que tenemos claro como es una dirección IPv6 podemos entrar a valorar las diferentes partes internas que tiene. Las direcciones van acompañadas de una barra invertida seguida de una cifra, por ejemplo /64. Esto indica que en la dirección los primeros 64 bits corresponden a

la red mientras que los bits restantes, 64, corresponden al espacio de hosts. Con la Ilustración 8 se da un ejemplo:

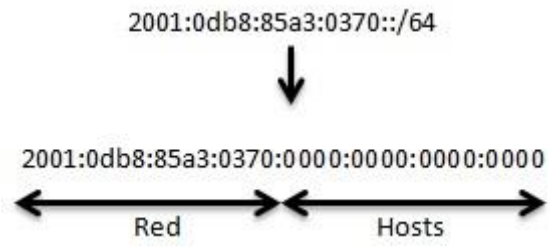

Ilustración 8: Ejemplo empleo barra invertida

A continuación vamos a entrar en los diferentes tipos de redes para poder ver de forma detallada los diferentes formatos. Existen tres tipos de direcciones:

- Unicast: la dirección está asociada a una interfaz concreta.
- Anycast: la dirección está asociada a un grupo de interfaces, normalmente pertenecientes a varios nodos. Un paquete enviado a una de estas direcciones suele ir destinado a la interfaz más cercana, siendo la cercanía medida mediante la métrica del protocolo de enrutamiento.
- Multicast: la dirección está asociada a un grupo de interfaces. Cuando se envía un mensaje a una de estas direcciones el mensaje llega a todas las interfaces identificadas por la dirección.

Las direcciones *broadcasts* de IPv4 son sustituidos por las direcciones *multicast*.

A continuación, para mostrar el formato de las diferentes direcciones se utilizarán tablas. Las tablas indican los 128 bits que forman cada dirección. A continuación, la dirección en su totalidad con los 128 bits:

128 bits
Dirección de nodo

Tabla 2: Dirección IPv6 básica

A continuación se verán las diferentes direcciones IPv6, las cuales tendrán diferentes partes, y gracias a la tabla se podrán ver esas partes junto con el tamaño que tienen.

2.5.3.1 Unicast

Hay varios tipos de direcciones unicast: *global unicast, site-local unicast y link-local unicast*. Las direcciones tienen la siguiente forma (aunque la dirección del nodo puede mostrarse de forma más detallada):

n bits	128-n bits
Prefijo de subred	Identificador de interfaz

Tabla 3: Dirección IPv6

Los identificadores de interfaz se utilizan en IPv6 para identificar interfaces en un enlace. Es necesario que sean únicos dentro de un prefijo de subred. También se recomienda que el mismo identificador no se asigne a varios nodos dentro de un enlace. Hay que destacar que el hecho de que un identificador sea único es independiente del hecho de que una dirección IPv6 lo sea. Es necesario que el identificador de interfaz tenga un tamaño de 64 bits y esté formado mediante el EUI-64 Modificado, excepto si comienza con el valor binario 000.

2.5.3.1.1 Global UnicastAddress

Algunas dirección unicast globales son las direcciones IPv6 que contienen direcciones IPv4 embebidas.

n bits	m bits	128-n-m
Prefijo global de enrutamiento	Identificador de subred	Identificador de interfaz

Tabla 4: Dirección unicast global

El valor del prefijo de ruta global es un valor asignado a un sitio, el id de subred es un identificador de un enlace dentro del sitio y el identificador de interfaz ya ha sido explicado.

2.5.3.2Enlace-local

Una dirección de enlace-local es una dirección unicast que utiliza un valor específico para el prefijo de red.

10 bits	54 bits	64 bits
Prefijo	Ceros	Identificador de interfaz

Tabla 5: Dirección de enlace local

2.5.3.3Dirección indefinida

Es la dirección que tiene todos valores a cero, 0:0:0:0:0:0:0:0. No debe asignarse a un nodo ni debe usarse de forma normal como parte de un paquete o de una cabecera. Indica la ausencia de dirección.

Se usa cuando un nodo inicializado aún no tiene su propia dirección y tiene que enviar menajes, se le da este valor al campo de la dirección de origen.

2.5.3.4Dirección *loopback*

Cuando un nodo quiere enviarse un mensaje a si mismo utiliza la dirección de *loopback*, que se corresponde con 0:0:0:0:0:0:0:1. No debe asignarse a ninguna interfaz física ni debe incluirse como dirección de origen en paquetes que se envíen fuera de un nodo. Una interfaz que recibe un paquete con una dirección de destino de *loopback* debe descartarlo.

2.5.3.5Anycast

Como vimos en la breve descripción inicial, una dirección *anycast* viene asociada a varias interfaces, las cuales normalmente estarán en diferentes nodos. Cuando se envía un mensaje a una dirección *anycast* el mensaje termina en la interfaz que esté más cerca de la dirección, de acuerdo a la métrica que emplee el protocolo de enrutamiento.

Utilizan el mismo espacio de direcciones que las *unicast*, por tanto podemos ver que de forma sintáctica son indistinguibles. Cuando una dirección *unicast* es asignada a varias interfaces, la dirección se convierte en *anycast*, y los nodos a los que pertenecen dichas interfaces deben ser configurados de tal manera que sepan que es una dirección *anycast*.

Por cada dirección *anycast* asignada existe un prefijo P de esa red que identifica la región topológica a la que pertenecen todas las direcciones *anycast* residente. Dentro de la región identificada por P, la dirección *anycast* debe ser mantenida como una entrada separada en el sistema de enrutamiento; fuera de la región identificada por P, la dirección *anycast* puede ser agregada dentro de la entrada de ruta por el prefijo P.

Un uso de las direcciones *anycast* es identificar el set de *routers* que pertenecen a una organización proveedora de servicios de internet. Otros posibles usos son la identificación del set de *routers* unidos a una subred particular, o del conjunto de *routers* que proporcionan entrada en un dominio de enrutamiento particular.

n bits	128-n bits
Prefijo de subred	00000000000000

Tabla 6: Dirección Anycast

2.5.3.6Multicast

Una dirección multicast es un identificador para un grupo de interfaces. Una interfaz puede pertenecer a cualquier número de grupos multicast. Tienen el siguiente formato:

8 bits	4 bits	4 bits	112 bits
Prefijo	Flgs	Scop	ID de grupo
11111111	0RPT	XXXX	

Tabla 7: Dirección Multicast

Los primeros 8 bits identifican la dirección como multicast. El campo flgs dispone de 4 flags:

- El flag de mayor orden está reservada para uso futuro y es inicializada a cero.
- El flag R tiene los siguientes valores:
  - R=0 el punto de Rendezvous no está integrado.
  - R=1 el punto de Rendezvous está integrado.
- El flag P tiene los siguientes valores:
  - P=0 indica que es una dirección sin información de prefijo.
  - P=1 indica que es una dirección basada en el prefijo de red.
- El flag T tiene los siguientes valores:
  - T=0 indica que la dirección multicast es mundialmente válida.
  - T=1 indica que es una dirección multicast asignada de forma dinámica.
- Scop, de scope que significa ámbito, es un campo de 4 bits usado para indicar donde es válida y única la dirección. Puede tomar los siguientes valores:

Valor	Ámbito	Descripción
0x0	Reservado	
0x1	Interface-local:	Solo abarca un interfaz de un nodo, solo es útil para la transmisión loopback del tráfico multicast.
0x2	Link-local	Los ámbitos de enlace local y site-local abarcan las mismas regiones que los ámbitos unicast correspondientes.
0x2	Reservado	
0x4	Admin-local	Es el ámbito más pequeño que debe ser configurado manualmente, es decir, no deriva automáticamente e la conexión física sin relación alguna con multicast.
0x5	Site-local	Los ámbitos de enlace local y site-local abarcan las mismas regiones que los ámbitos unicast correspondientes.
0x6	Sin asignar	
0x7	Sin asignar	
0x8	Organization-local	Abarca múltiples ubicaciones que pertenecen a la misma organización
0x9	Sin asignar	
0xa	Sin asignar	
0xb	Sin asignar	
0xc	Sin asignar	
0xd	Sin asignar	
0xe	Global	
0xf	Reservado	

Tabla 8: valores del ámbito

Los ámbitos que están sin asignar están disponibles para que los administradores definan regiones multicast adicionales.

2.5.3.7Direcciones especiales:

Existen una serie de direcciones especiales que se muestran a continuación:

2.5.3.7.1 Ruta por defecto

- ::/0

Es la ruta por defecto para el tráfico unicast.

2.5.3.7.2 Direcciones locales:

- fe80::/10

Las direcciones de enlace local son válidas y únicas en la red local. Dentro de este rango solo se usa una subred(54 bits a cero), generando un formato eficaz de fe80::/64. Son como las direcciones de auto-configuración de IPv4 169.254.0.0/16.

- fe80::/10

Direcciones locales únicas, usadas para comunicaciones locales. Son como las direcciones privadas IPv4 (10.0.0.0/8, 172.16.0.0/12 y 192.168.0.0/16). Incluyen una secuencia pseudoaleatoria en el prefijo de enrutamiento para minimizar el riesgo de conflictos en la interconexión de plataformas diferentes. A pesar de ser de uso local son de ámbito global, es decir, se espera que sean únicas en todo el mundo.

2.5.3.7.3 Uso especial:

Algunos grupos de direcciones tienen usos específicos, como por ejemplo:

- 2001::/32

Protocolo de túneles Teredo.

- 2001:2::/48

Asignado al BMWG para comparativas en ipv6.

- 2001:10;::/28

Direcciones ipv6 no enrutables usadas para identificadores criptográficos hash.

2.5.3.7.4 Documentación:

- 2001:db8::/32

Se utilizan para escribir modelos de direcciones ipv6 o plasmar modelos de red. Similar a la red 192.0.2.0/24

2.6 6LoWPAN

6LoWPAN[14] es el acrónimo de *IPv6 over Low Power Wireless Personal Area Network*, que utiliza el estándar IPv6 sobre el estándar 802.15.4. La principal ventaja que aporta frente a otro tipo de estándares que funcionan sobre 802.15.4 es que permite utilizar la infraestructura IP para redes de comunicación.

Por otro lado, hay que tener en cuenta que el estándar 802.15.4 tiene ciertas restricciones: tamaño de paquetes reducido, poco ancho de banda o poca potencia de batería entre otras. Lo primero que chocará es ver que existe una restricción del tamaño de los paquetes, a 127 octetos, cuando el tamaño de los paquetes IPv6 es muy superior, al menos diez veces mayor.

A continuación, entraremos en detalle con los diferentes aspectos de 6LoWPAN mediante un repaso de las diferentes capas que lo forman.

2.6.1 Capas 6LoWPAN

6LoWPAN utiliza un modelo de capas similar a OSI, con algunos pequeños cambios. En este apartado se explicará de forma detallada cada una de ellas y su relación con TinyOS, ya que será el sistema operativo elegido para realizar el trabajo.

La Ilustración 9 muestra una comparación entre los tres modelos de pila: TCP/IP, OSI y la pila de 6LoWPAN. Lo primero que vemos es la similitud entre el modelo OSI y el modelo 6LoWPAN, donde la principal diferencia es que dispone de una capa adicional, la de aplicación, y que hay dos capas, las capas de sesión y presentación en el modelo OSI, que no se utilizan en 6LoWPAN. Debido a esto, nos centraremos en las otras seis capas (de abajo a arriba): física, enlace de datos, adaptación, red, transporte y aplicación.

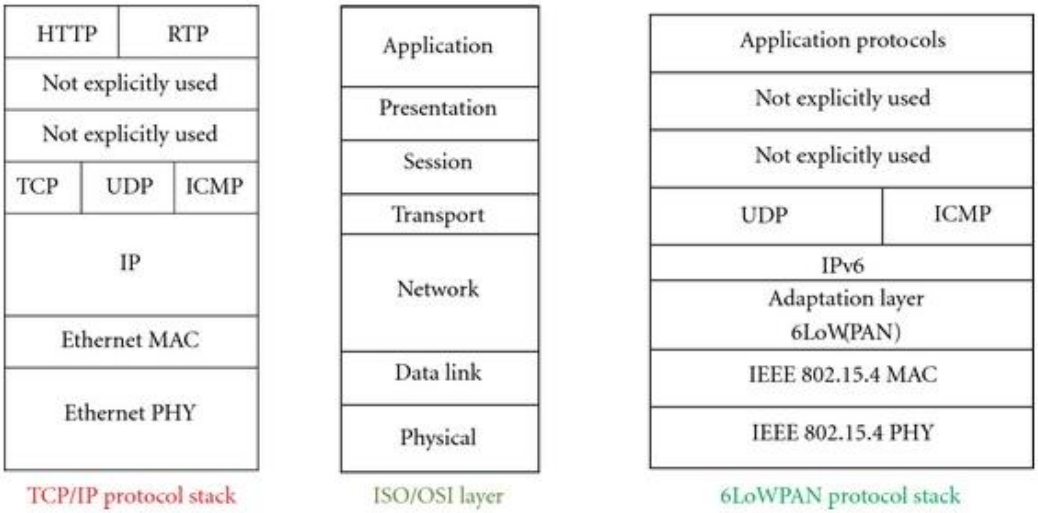

Ilustración 9: Modelos de capas

2.6.1.1 Capa Física

La capa física permite la transmisión y recepción de paquetes de a través del medio físico. Esta capa está basada íntegramente en la capa física del estándar 802.15.4, por tanto, se mantiene la velocidad de envío de datos en 250 Kbps y la frecuencia de operación entre 2400 y 2483,5 MHz.

Además, provee un servicio de gestión de interfaz que permite acceder al gestor de funciones de las capas y mantener una base de datos con la información relacionada a las redes de área personal (PAN).

En la Ilustración 10 se pueden ver los campos que forman el paquete de la capa física. Después, se explicarán los diferentes campos de forma individual.

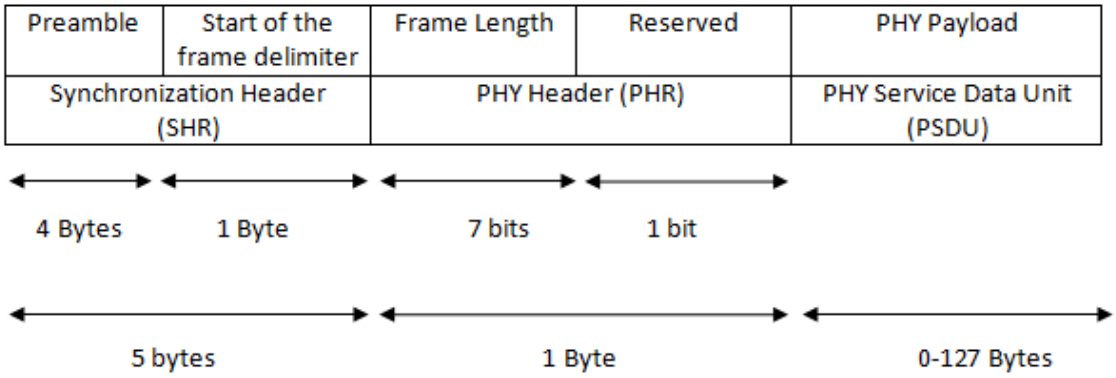

Ilustración 10: Paquete capa física

La capa física se divide en: SHR (*Synchronization header*), PHR (*Physical header*) y PSDU (*Physical service data unit*).

- SHR: la cabecera de sincronización, que se divide en dos campos de 4 y 1 bytes. El campo de 4 bytes es el preámbulo de secuencia (Preamble) y el de 1 byte es el inicio del delimitador de trama (Start of FrameDelimiter).
- PHR: la cabecera física, un campo de un byte con la longitud de la trama.
- PSDU: la unidad de datos de servicio de la capa física, es un campo variable que puede llegar hasta los 127 bytes. En este campo se incluye la carga.

2.6.1.2 Capa de Enlace de Datos

La capa de enlace es conocida como la subcapa MAC. Permite que exista comunicación de un solo salto entre dispositivos. Esto se consigue mediante la transmisión y recibimiento de unidades de datos del protocolo MAC (MPDU). Al igual que en la capa anterior, está basada de forma íntegra en la capa MAC del estándar 802.15.4.

Además, incluye un servicio de interfaz de gestión para la entidad de gestión de la subcapa MAC.

Se definen cuatro tipos de tramas:

- Data frame: se utilizan en las transferencias de datos.
- Beaconframe: son tramas generadas por un coordinador y se emplean para llevar a cabo la sincronización.
- Commandframe: son usados por la entidad de gestión de la subcapa.
- Acknowlegdementframe: sirven para aceptar las recepciones exitosas de las tramas.

A continuación, en la Ilustración 11, se muestra el formato del paquete MAC seguido de la descripción de los diferentes campos que lo forman.

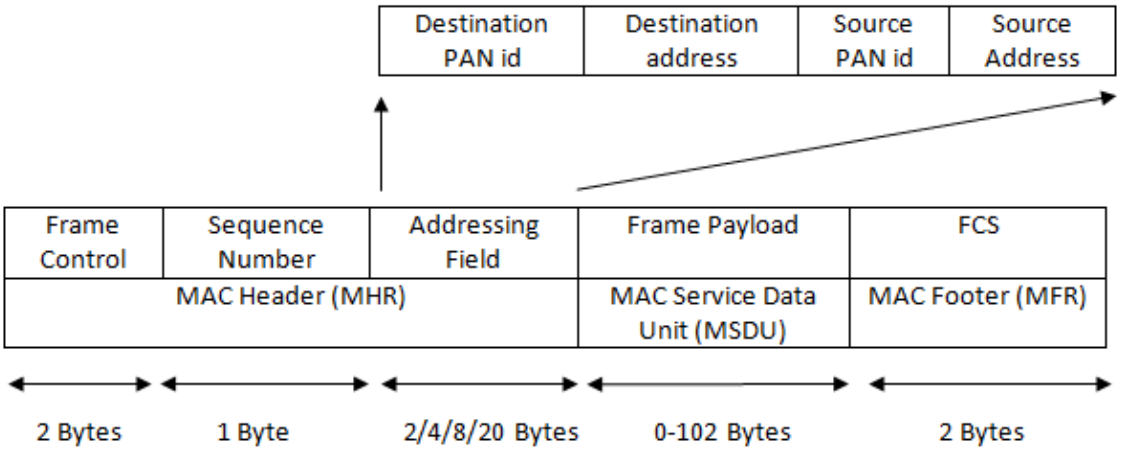

Ilustración 11: paquete MAC

La subcapa MAC se puede dividir en tres partes: MHR (*MAC header*), MSDU (*MAC service data unit*) y MFR (*MAC footer*).

- MHR: corresponde a la cabecera MAC, donde tiene un campo de control (Frame Control) que especifica el tipo de trama, con un tamaño de 2 bytes. También dispone de un campo de 1 byte que verifica la integridad de la trama (Data Sequence Number). Por último, encontramos el campo encargado de la información de la dirección (Addressing Field) cuya longitud puede variar entre los 2 y los 20 bytes.
- MSDU: es la unidad de datos de servicio MAC. Es un campo variable que puede llegar hasta los 102 bytes. En este campo se incluye la carga.
- MFR: el final de trama MAC, es el encargado del chequeo de la trama y tiene un tamaño de 2 bytes.

Por último, mencionar que esta capa dispone de algunos mecanismos para el control de la congestión y las colisiones. En algunos casos, donde la solución a estos problemas pasa por el empleo de métodos entre capas, *cross-layer* en inglés, trabaja junto a la capa de red.

Los principales métodos empleados son CSMA/CD (C*arried Sense Multiple Access Collision Detect*) [15], CDMA (*Code Division Multiple Access*)o TDMA (*Time Division Multiple Access*).

2.6.1.3 Capa de adaptación

Esta capa es completamente nueva, no está incluido ni el modelo de capas TCP/IP ni en el modelo de capas OSI. Es el componente principal de 6lowpan y cumple con las siguientes funciones:

- **Compresión de cabeceras:** el tamaño máximo de los paquetes de estándar 802.15.4 es de 127 bytes. Debido a los tamaños de las diferentes cabeceras de TCP/IP,(IPv6 son 40 bytes, UDP e ICMP son 4 bytes cada una y TCP son 20 bytes), se puede ver que sin compresión el estándar 802.15.4 no será capaz de enviar carga útil dentro de los paquetes, lo cual hará que sea inútil mandarlos. Es una parte fundamental ya que es lo que permite que se haya podido adaptar IPv6 al estándar 802.15.4, pero es también

un tema muy técnico que no tiene relación directa con la finalidad de este trabajo, por ello, no vamos a entrar en los detalles de este tema.

- **Fragmentación y reensamblado de paquetes:** debido al tamaño de la MTU de IPv6, 1280 bytes, es imposible que se pueda enviar todo encapsulado en una trama 802.15.4. El MTU de 802.15.4, como vimos antes, es de 127 bytes. A esto hay que descontarle los 25 bytes de cabeceras y los bytes de seguridad, que serán 21 por ser la opción impuesta, dejando el tamaño de la carga útil en 81 bytes. Así, serán necesarios al menos 16 paquetes 802.15.4 para poder llevar un paquete IPv6 de un punto a otro.
- **Enrutamiento:** los nodos del borde de la WSN deben ser capaces de enrutar los paquetes IPv6 dentro de los nodos de la WSN desde fuera y enrutar los paquetes de la red fuera de la red IP. Para ello existen diferentes protocolos de enrutamiento, los cuales se usaran en función del lugar donde se necesiten enrutar lo paquetes. Se diferenciará el enrutamiento dentro del espacio PAN con el enrutamiento de paquetes entre el espacio PAN y el exterior de la red, el dominio IPv6. Para enrutar los paquetes dentro del espacio PAN se utilizará el enrutamiento en malla (*mesh-routing*). A continuación, se muestran algunos de los protocolos de enrutamiento:
    - **LOAD (*6LoWPAN Ad-hoc RoutingProtocol*):** es una simplificación de *Ad-hoc On-Demand Distance Vector* (AODV) para 6lowpan. Es un protocolo de enrutamiento desarrollado para redes móviles ad-hoc. Permite tanto unicast como multicast y es un algoritmo "On-Demand" lo que hace que construya rutas entre nodos cuando son necesitadas por el origen. Mantienen esas rutas mientras son necesarias.
    - **DYMO-Low (*Dynamic MANET On-Demand*):** desarrollado para redes de nodos móviles en redes de múltiples saltos sin cables. Ofrece adaptación a los cambios de topología de la red y determina rutas unicast entre nodos de una red.
    - **Hi-Low (*Hierarchicalrouting*):** basado en las direcciones de 16 bits de 6lowpan.

Además, existen otras funciones que puede implementar esta capa, como el descubrimiento de vecinos o el soporte de direcciones multicast.

2.6.1.4 Capa de red

En esta capa se realizan varias funciones. Primero, se cumplen las necesidades que tiene el direccionamiento en IPv6. Después, se realiza el enrutamiento y el mapeado. Por último, se ven los servicios de seguridad apropiados.

Existen dos formas de considerar el enrutamiento: *mesh-under* y *route-over*.

En una organización *mesh-under*, la pila de red no realiza ningún enrutamiento IP dentro de la LoWPAN; en su lugar, la capa de adaptación busca suplir la falta de un broadcast completo a nivel físico mediante el enrutado y reenvio de forma transparente de paquetes dentro de la LoWPAN. Aquí se presenta un desafío, ya que emular el enlace lógico es más complejo en una LoWPAN que en una infraestructura tradicional basado en el estándar 802.11. Las topologías *mesh* requieren que se reenvíen los paquetes a través de múltiples saltos, mientras que en el

caso del multicast local el enlace debe entregar el paquete a todos los nodos que pertenecen a la LoWPAN. Muchos mecanismos que existen para formar, mantener y diagnosticar rutas IP deben ser recreados a nivel de enlace para poder funcionar de forma estable.

De forma alternativa, *route-over* realiza el enrutamiento en la capa IP, con cada nodo sirviendo como un *router* IP. Podemos verlo como si fuera una colección de superposiciones de ámbitos de enlace local, donde cada dominio de enlace local está definido por la conectividad inherente del canal radio. A diferencia de *mesh-under*, *route-over* soporta los mecanismos de reenvío de la capa tres dentro de LoWPAN que puede utilizar las capacidades de la capa de red definidas por IP, como el enrutamiento IPv6 e ICMPv6 para gestión y configuración. *Route-over* también permite a los protocolos de enrutamiento IP abarcar diferentes tecnologías de enlace, habilitando una mejor integración en redes más capaces. También permite a los protocolos basados en IP restringir la comunicación en la radio local, en lugar de en toda la LoWPAN. Los problemas entre la capa de enlace y la capa de red no son únicos de 6lowpan, surgen en redes FrameRealy, Asynchronous Transfer Mode (ATM), switched Ethernet, y 802.11 meshing.

La principal diferencia entre los dos métodos no radica en la fase de establecimiento de ruta sino en el proceso de reenvío de paquetes/fragmentos. En route-over cada salto de capa de enlace es un salto IP y cada nodo actúa como un *router* IP. El paquete es reenviado salto a salto desde el origen al destino entre estos enlaces. La carga útil del paquete esta encapsulada en cabeceras IP.

Después, el paquete IP es fragmentado y cada fragmento es enviado al siguiente salto según la información de la tabla de enrutamiento. Si la capa de adaptación recibe todos los fragmentos satisfactoriamente crea un paquete IP y lo manda a la capa de red. Después, envía ese paquete a la capa superior, capa de transporte, si era el destino del paquete. De otra forma, envía el paquete al siguiente salto según la información de la tabla de enrutamiento.

2.6.1.5 Capa de Transporte

Se encarga de entregar los datos al proceso de aplicación apropiado en el PC del host. Existen dos tipos de protocolo de transporte: TCP (*Transmission Control Protocol*) y UDP (*User Datagram Protocol*). En el origen, la conexión con el protocolo se basa en la aplicación. De este modo se crean cualquiera de los procesos TCP o UDP. Los datos de la capa de aplicación se organizan en segmentos TCP o UDP y se adjuntan al proceso creado. En el destino, después de recibir el segmento de datos de la capa de red, la capa de transporte procesa el segmento basado en el protocolo usado y lo envía a la capa de aplicación.

2.6.1.5 Comparativa entre TCP y UDP

El principal punto fuerte de TCP es la capacidad de enviar mensajes entre aplicaciones de forma fiable. Esto lo consigue mediante diferentes mecanismos con: establecimiento de conexión, *checksums* para evitar errores, números de secuencia para ordenar los segmentos de datos y temporizadores para evitar las pérdidas y poder reenviar mensajes.

El problema de TCP es que permitir todos estos puntos positivos tiene un coste en la complejidad y eficiencia del protocolo. Además, todos estos elementos ocupan espacio en los segmentos de datos, 20 bytes en la cabecera, por lo tanto se reduce el tamaño de la carga útil que se puede incluir en cada segmento y eso hace que los mensajes grandes deban ser fragmentados en una cantidad mayor de elementos que si estuviéramos usando UDP.

UDP no aporta garantías de entrega de datos, ya que no incluye apenas información adicional en los segmentos de datos. La cabecera de UDP solo ocupa 8 bytes, menos de la mitad que TCP. Si se quiere asegurar la entrega de los datos será necesario que se programen las soluciones en capas superiores. Es un protocolo muy utilizado para la transmisión de audio o de vídeo.

Una vez vistas las diferencias entre TCP y UDP desde el punto de vista de TCP/IP vamos a ver los motivos de elección dentro de 6LoWPAN.

Con 6lowpan se utiliza UDP debido a que en aspectos de rendimiento, eficiencia y complejidad es preferible a TCP. Además, actualmente la pila de TCP está en modo experimental y en ocasiones puede funcionar de manera anómala.

2.6.1.6 Capa de aplicación

La capa de aplicación usa una interfaz de socket para aplicaciones específicas. Cada aplicación 6lowpan abre un socket que es usado para enviar y recibir paquetes. Cada socket está asociado a un protocolo, TCP o UDP, un puerto de origen y un puerto de destino.

Algunos ejemplos de aplicaciones son la monitorización del entorno o el uso como dispositivos de seguridad.

3. Especificación de Requisitos

La toma de requisitos es una parte fundamental en cualquier proyecto, ya que es donde se plasma lo que el cliente quiere y es donde se tiene que detallar las características y funcionalidades del proyecto.

Aunque se puede profundizar de una manera muy amplia en este tema, al no ser esta una de las prioridades de este trabajo vamos a utilizar solo dos tipos de requisitos: requisitos de usuario y requisitos de sistema.

Los requisitos de usuario especifican la funcionalidad de la aplicación, las tareas que debe ser capaz de realizar. Los requisitos de sistema especifican cómo debe realizarse el producto, ya que recoge las restricciones que se le imponen.

En Tabla 9 se muestra la información que se recogerá en cada uno de los requisitos. Antes de entrar con los tipos de requisitos se explicará de forma detallada los diferentes campos que tiene la tabla, ya que al usar la misma para los dos casos solo es necesario explicarlo una vez.

Requisito-XXX	
Campo	**Valor**
Identificador	RXX-XX
Nombre	
Descripción	
Prioridad	Alta/Media/Baja
Fuente	
Verificabilidad	Alta/Media/Baja

Tabla 9: Ejemplo tabla de requisitos

- Identificador: identifica tanto el tipo de requisito como su número.
- Nombre: Breve descripción del requisito.
- Descripción: descripción detallada del requisito.
- Prioridad: utilizado para la planificación del desarrollo.
- Fuente: indica la procedencia del requisito, ya sea cliente o desarrollador.
- Verificabilidad: faciliad para comprobar si e requisito se ha incorporado.

Los valores que puede tener el campo identificador son: Requisito-UXX o Requisito-RXX, donde U se refiere a los requisitos de usuario, la S a los de sistema y XX se refiere a la numeración que se aplicará en cada uno de los requisitos, comenzando por 01.

A parte de estos campos sería posible incluir otros campos, como puedan ser claridad, estabilidad o necesidad. En este caso, se ha decidido no inlcuirlos ya que no ofrecen información relevante. La necesidad, por ejemplo, siempre será alta en todos los requisitos, ya

que todos tienen una importancia similar, además de no existir requisitos que puedan ser opcionales.

3.1 Aproximación al problema

Antes de entrar en la descripción de los requisitos se van a realizar una breve aproximación para ayudar a la comprensión del problema a resolver, ya que en caso contrario podría ser una lectura confusa. Debe quedar claro que no se mencionarán todos los requisitos incluidos en los puntos 3.2 y 3.3, solo algunos que nos permitan hacernos una idea de lo que contiene cada apartado.

Como ya sabemos gracias a la introducción, el prototipo debe encargarse de que uno o varios nodos tomen lecturas y las envíen a una estación base utilizando para ello el protocolo de comunicación 6lowpan.

En el apartado 3.2 se encuentran los requisitos que se encargan de especificar los elementos que pueden comunicarse con otros, así como los datos que deben tomarse en los nodos sensores y cada cuanto tiempo deben tomarse. Por último se especificarán los elementos que hacen referencia al almacenamiento de datos.

En el apartado 3.3 Especificación de requisitos de sistema se incluyen los requisitos que hacen referencia a cómo deben ser los diferentes elementos que están involucrados en el trabajo. Lo primero que encontraremos serán las versiones de los lenguajes de programación o del sistema operativo. También se especificarán los modelos de las diferentes partes del prototipo y los estándares de comunicación y transporte.

3.2 Especificación de requisitos de usuario

A continuación se muestran los requisitos que hacen referencia a las funcionalidades del prototipo:

Requisito-U01	
Campo	**Valor**
Identificador	RUC-01
Nombre	Prototipo.
Descripción	Se construirá un prototipo que permita evaluar el protocolo 6LoWPAN sobre una WSN. La WSN estará formada por un número variable de nodos sensores, un *gateway* y una estación base.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 10: RUC-01

Requisito-U02	
Campo	**Valor**
Identificador	RUC-02
Nombre	Envío datos nodo sensor a estación base.
Descripción	El nodo sensor podrá enviar los datos a la estación base.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 11: RUC-02

Requisito-U03	
Campo	**Valor**
Identificador	RUC-03
Nombre	Envío datos nodo sensor a nodo sensor.
Descripción	El nodo sensor podrá enviar los datos a otro nodo sensor.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 12: RUC-03

Requisito-U04	
Campo	**Valor**
Identificador	RUC-04
Nombre	Recepción mensajes nodo sensor.
Descripción	El nodo sensor podrá recibir mensajes de otro nodo.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 13: RUC-04

Requisito-U05	
Campo	**Valor**
Identificador	RUC-05
Nombre	Nodo *gateway* – recibir mensajes nodos.
Descripción	El nodo *gateway* podrá recibir mensajes desde los nodos sensores.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 14: RUC-05

Requisito-U06	
Campo	**Valor**
Identificador	RUC-06
Nombre	Nodo *gateway* – enviar mensajes estación base.
Descripción	El nodo *gateway* podrá enviar mensajes a la estación base.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 15: RUC-06

Requisito-U07	
Campo	**Valor**
Identificador	RUC-07
Nombre	Estación base – recibir mensajes.
Descripción	La estación base podrá recibir mensajes del nodo *gateway*.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 16: RUC-07

Requisito-U08	
Campo	**Valor**
Identificador	RUC-08
Nombre	Obtener temperatura.
Descripción	El nodo sensor obtendrá los datos de temperatura con la placa de sensores.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 17: RUC-08

Requisito-U09	
Campo	**Valor**
Identificador	RUC-09
Nombre	Dirección de destino.
Descripción	Para enviar las mediciones será necesario establecer una dirección de destino en el nodo sensor.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 18: RUC-09

Requisito-U10	
Campo	**Valor**
Identificador	RUC-10
Nombre	Puerto de destino.
Descripción	Para enviar las mediciones será necesario establecer un puerto de destino en el nodo sensor.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 19: RUC-10

Requisito-U11	
Campo	**Valor**
Identificador	RUC-11
Nombre	Envío estadísticas – IP
Descripción	El nodo sensor obtendrá datos estadísticos referentes al protocolo IP mediante los siguientes campos: ip.sent, ip.forwarded, ip.rx_drop, ip.tx_drop, ip.fw_drop, ip.rx_total, ip.real_drop, ip.hlim_drop, ip.sendDone_el, ip.fragpool, ip.sendinfo, ip.sendentry, ip.sndqueue, ip.encfail e ip.heapfree.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 20: RUC-11

Requisito-U12	
Campo	**Valor**
Identificador	RUC-12
Nombre	Envío estadísticas – UDP
Descripción	El nodo sensor obtendrá datos estadísticos referentes al protocolo UDP mediante los siguientes campos: udp.total, udp.failed, udp.seqno, udp.sender.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 21: RUC-12

Requisito-U13	
Campo	**Valor**
Identificador	RUC-13
Nombre	Envío estadísticas – ICMP
Descripción	El nodo sensor obtendrá datos estadísticos referentes al protocolo ICMP mediante el siguiente campo: icmp.rx.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 22: RUC-13

Requisito-U14	
Campo	**Valor**
Identificador	RUC-14
Nombre	Envío estadísticas – ruta.
Descripción	El nodo sensor obtendrá datos estadísticos referentes a la ruta que toman los mensajes mediante los siguientes campos: route.hop_limit, route.parent, route.parent_metric, route.parent_etx.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 23: RUC-14

Requisito-U15	
Campo	**Valor**
Identificador	RUC-15
Nombre	Temporizador datos temperatura.
Descripción	El nodo sensor tomará mediciones del sensor una vez por minuto.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 24: RUC-15

Requisito-U16	
Campo	**Valor**
Identificador	RUC-16
Nombre	Temporizador datos estadísticas.
Descripción	El nodo sensor tomará mediciones de los sensores cada 10 minutos.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 25: RUC-16

Requisito-U17	
Campo	**Valor**
Identificador	RUC-17
Nombre	Placa de sensores.
Descripción	El nodo sensor usará una placa de sensores para tomar las mediciones.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 26: RUC-17

Requisito-U18	
Campo	**Valor**
Identificador	RUC-18
Nombre	Elementos nodo *gateway*
Descripción	El nodo *gateway* estará formado por un nodo sensor unido a un dispositivo *gateway*.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 27: RUC-18

Requisito-U19	
Campo	**Valor**
Identificador	RUC-19
Nombre	Conexión *gateway*
Descripción	El nodo *gateway* estará conectado a la estación base vía USB.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 28: RUC-19

Requisito-U20	
Campo	**Valor**
Identificador	RUC-20
Nombre	Almacenamiento.
Descripción	La estación base almacenará los datos de los dos tipos de mensaje en una base de datos.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 29: RUC-20

Requisito-U21	
Campo	**Valor**
Identificador	RUC-21
Nombre	Base de datos – tabla temperatura.
Descripción	La base de datos tendrá una tabla para los datos de temperatura con los siguientes campos: marca de tiempo, contador de mensajes, identificador del nodo emisor, dato de temperatura del sensor, temperatura en grados Celsius.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 30: RUC-21

Requisito-U22	
Campo	**Valor**
Identificador	RUC-22
Nombre	Base de datos – tabla estadísticas.
Descripción	La base de datos tendrá una tabla para los datos de estadísticas con los siguientes campos: ip.sent, ip.forwarded, ip.rx_drop, ip.tx_drop, ip.fw_drop, ip.rx_total, ip.real_drop, ip.hlim_drop, ip.sendDone_el, ip.fragpool, ip.sendinfo, ip.sendentry, ip.sndqueue, ip.encfail, ip.heapfree, udp.total, udp.failed, udp.seqno, udp.sender, icmp.rx, route.hop_limit, route.parent, route.parent_metric y route.parent_etx.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 31: RUC-22

3.3 Especificación de requisitos de sistema

A continuación los requisitos que especifican cómo deben ser los elementos que forman parte del prototipo:

Requisito-S01	
Campo	**Valor**
Identificador	RUS-01
Nombre	Versión de TinyOS.
Descripción	La versión de TinyOS será la 2.1.1.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 32: RUS-01

Requisito-S02	
Campo	**Valor**
Identificador	RUS-02
Nombre	Versión de nesC.
Descripción	La versión de nesC será la versión 1.3.0.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 33: RUS-02

Requisito-S03	
Campo	**Valor**
Identificador	RUS-03
Nombre	Versión de Python.
Descripción	La versión de Python será la versión 2.4.0 o una superior.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 34: RUS-03

Requisito-S04	
Campo	**Valor**
Identificador	RUS-04
Nombre	Versión de Linux.
Descripción	La versión del sistema operativo Linux será la versión 10.4 o superior.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 35: RUS-04

Requisito-S05	
Campo	**Valor**
Identificador	RUS-05
Nombre	Versión de MySQL.
Descripción	La versión de MySQL será la 5.5.32 o superior.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 36: RUS-05

Requisito-S06	
Campo	**Valor**
Identificador	RUS-06
Nombre	Modelo sensores de la red.
Descripción	El modelo de los sensores de la red será MicaZ.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 37: RUS-06

Requisito-S07	
Campo	**Valor**
Identificador	RUS-07
Nombre	Modelo nodo *gateway*.
Descripción	El modelo del nodo *gateway* será Mib520.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 38: RUS-07

Requisito-S08	
Campo	**Valor**
Identificador	RUS-08
Nombre	Modelo placa de sensores.
Descripción	El modelo de la placa de sensores será MTS300.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 39: RUS-08

Requisito-S09	
Campo	**Valor**
Identificador	RUS-09
Nombre	Estándar comunicación.
Descripción	Se empleará el estándar 6lowpan para realizar las comunicaciones entre la WSN y la estación base.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 40: RUS-09

Requisito-S10	
Campo	**Valor**
Identificador	RUS-10
Nombre	Protocolo capa de transporte - UDP
Descripción	Podrá utilizarse UDP como protocolo de la capa de transporte.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 41: RUS-10

Requisito-S11	
Campo	**Valor**
Identificador	RUS-11
Nombre	Simulación.
Descripción	Si se necesitara realizar simulaciones del código de los sensores se empleará el simulador TOSSIM.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 42: RUS-11

Requisito-S12	
Campo	**Valor**
Identificador	RUS-12
Nombre	WSN – Estación base
Descripción	La WSN tendrá una estación base.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 43: RUS-12

Requisito-S13	
Campo	**Valor**
Identificador	RUS-13
Nombre	WSN – nodo *gateway*
Descripción	La WSN tendrá un nodo *gateway*.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 44: RUS-13

Requisito-S14	
Campo	**Valor**
Identificador	RUS-14
Nombre	WSN – nodo sensor
Descripción	La WSN tendrá al menos un nodo sensor.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 45: RUS-14

Requisito-S15	
Campo	**Valor**
Identificador	RUS-15
Nombre	Estación base.
Descripción	La estación base será un PC con acceso a internet.
Prioridad	**Alta**/Media/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 46: RUS-15

Requisito-U16	
Campo	**Valor**
Identificador	RUC-16
Nombre	Valor medición
Descripción	El valor de las mediciones será numérico y admitirá hasta dos dígitos decimales.
Prioridad	Alta/**Media**/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 47: RUS-16

Requisito-U17	
Campo	**Valor**
Identificador	RUC-17
Nombre	Formato medición
Descripción	En nesC el formato del campo de la medición será uint16_t.
Prioridad	Alta/**Media**/Baja
Fuente	Cliente.
Verificabilidad	**Alta**/Media/Baja

Tabla 48: RUS-17

4. Diseño

En capítulos anteriores, más concretamente en el capítulo Estado de la Cuestión, se ha visto que existen varias opciones para los distintos elementos que forman una red de sensores, como son los nodos sensores o el entorno de trabajo en el cual implementar el código.

Como nota inicial, debe quedar claro que algunos elementos son impuestos como requisitos de usuario, tal y como se describe en el capítulo anterior. En este capítulo se valoran las distintas opciones y se describen las decisiones de diseño tomadas.

Después, se explicará el diseño en sí del prototipo, donde se verá cada parte por separado y en profundidad.

4.1 Decisiones de diseño

A continuación se verán las elementos que presentaban varias opciones y se explicarán los motivos que nos han llevado a seleccionarlos.

4.1.1 Nodo sensor

En el capítulo Estado de la Cuestión se describió la estructura de una red de sensores, la cual debe estar necesariamente compuesta por al menos un nodo sensor y un nodo especial denominado *gateway*. En este capítulo se presentan los componentes básicos y elementos opcionales que se pueden incluir. En este apartado veremos una descripción más avanzada de los elementos físicos de los nodos sensores, y una explicación de los motivos para la elección del modelo.

A continuación se muestra una tabla con diferentes cualidades de los principales nodos sensores. Se incluyen las características físicas, del microprocesador, de la memoria, del transceptor y del sistema operativo que usan.

Sensores			
Características	MicaZ	TelosB	Iris
Tamaño (mm)[1]	58x32x7	65x31x6	58x32x7
Peso (g)	18	23	18
Micro controlador	ATMEGA 128	MSP430	ATMEGA 1281
Bus	8-bit	16-bit	8-bit
Reloj	8 MHz	4-8 MHz	8 MHz
Memoria RAM	4 KB	10 KB	8 KB

[1] Los tres valores del campo tamaño se corresponden con el largo, el ancho y el alto del sensor

Memoria Flash	512 KB	1024 KB	512 KB
Memoria EEPROM	128 KB	48 KB	128 KB
Radio	CC2420	CC2420	Atmel AT86RF230
Sistema Operativo	TinyOS, SOS	TinyOS, Contiki	TinyOS, Mote Runner

Tabla 49: Comparativa nodos sensores

De entrada se puede apreciar que los modelos MicaZ e Iris son casi idénticos, ya que solo se diferencian en la memoria RAM y la radio. En cambio, el modelo TelosB dispone de características diferentes a esos dos modelos, siendo ligeramente más grande y pesado y disponiendo de unos valores de memoria diferentes.

En este caso, la elección de los nodos sensores se ha realizado en base a la disponibilidad de modelos en el laboratorio. Así, el modelo disponible era MicaZ y es con el que se ha procedido a trabajar. En caso de haber tenido plena disposición de elección es probable que se hubiera elegido el modelo TelosB. Esto se debe a que disponen de un microprocesador superior que permite trabajar mejor con la última versión de TinyOS, donde abre un nuevo abanico de posibilidades.

4.1.2 Nodo Gateway

A continuación se muestran dos tipos de nodo *gateway*, uno con conexión USB y otro con la conexión Ethernet.

Gateway		
Características	MIB 520	MIB 600
Puerto de alimentación	USB	Ethernet
Puerto de datos	USB	Ethernet
Soporte a sensores	MicaZ/Mica2/Iris	MicaZ/Mica2/Iris

Tabla 50: Comparativa *gateway*

Primero, con los datos mostrados podemos ver que la diferencia entre los dos nodos *gateway* se centra en el modo de conexión. En el caso de MIB520 la comunicación entre el gateway y la estación base se realiza vía USB y en MIB600 se realiza por medio de una conexión Ethernet.

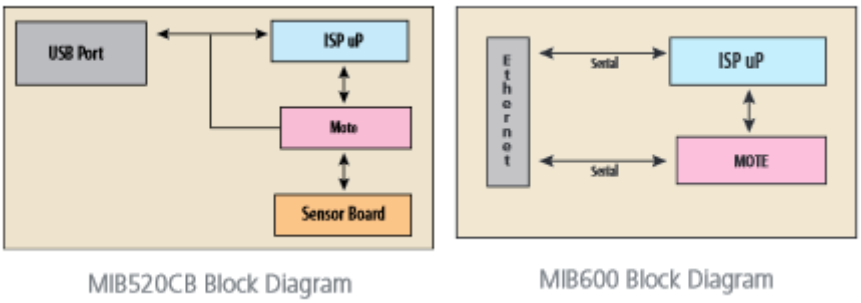

Ilustración 12: Diagrama nodos *gateway*

En la Ilustración 12 se puede ver que en el caso de la placa *gateway* MIB520 (izquierda en la ilustración), podemos incluir una placa de sensores mientras que en el caso del modelo MIB600 (derecha en la ilustración) no. Además, se puede conectar el nodo sensor directamente a la placa MIB520 mientras que en el caso de la placa MIB600 no, por tanto aquí tenemos suficientes elementos que hacen que nos decantemos por el modelo MIB520.

4.1.3 Placa de sensores

En el capítulo Estado de la Cuestión se habló de los complementos que pueden añadirse al nodo sensor, como por ejemplo la placa de sensores. Al igual que sucede con los elementos que hemos visto hasta ahora, existen multitud de placas que se pueden usar y que incluyen sensores de distintos tipos.

No todas las placas de sensores son compatibles con todos los nodos sensores. Para este trabajo se han considerado las placas de sensores MTS300 y MTS310, que son compatibles con MicaZ. Dentro de estos dos modelos de placas existen además los modelos CA y CB. La diferencia entre estos dos tipos reside en el tipo de sensor al que dan soporte; el modelo CA da soporte a sensores Mica y Mica2 mientras que el modelo CB da soporte a IRIS, MicaZ y Mica2. Además, el controlador de la señal es diferente en función del modelo que se use.

Una vez visto esto, queda claro que el modelo que elegiremos será el MTS3X0 CV, ya que es que dará soporte a los nodos sensores MicaZ. Ahora veremos la diferencia entre los modelos 300 y 310. Ambos modelos incluyen varios sensores en común: temperatura, luminosidad y sonido a través de un micrófono. En cambio, el modelo 310 incluye dos sensores nuevos: acelerómetro y magnetómetro. Como para este prototipo no nos vamos a centrar en la toma de múltiples datos ni los dos sensores adicionales del modelo 310 nos ofrecen nada especialmente interesante para nuestro trabajo, vamos a decantarnos por el modelo MTS300CB.

A continuación, en la Ilustración 13 se puede ver la placa de sensores elegida.

Ilustración 13: Placa de sensores MTS300CB

4.2 Diseño del prototipo

En este apartado se explicará todo lo necesario para comprender cómo se ha diseñado el prototipo de red de sensores con 6LoWPAN.

Primero, se especificará la forma de la red, donde se analizarán las diferentes partes que la componen y para qué sirve cada una de ellas.

4.2.1 Esquema del prototipo

La red de sensores estará formada por tres elementos: uno o varios nodos sensores, un nodo *gateway* y una estación base.

EL nodo sensor se encargará de tomar las lecturas de la placa de sensores, en nuestro caso solo se usará el sensor que mide la temperatura, y de tomar estadísticas correspondientes a los protocolos de red utilizados. Una vez ha tomado los datos el sensor envía el mensaje a la estación base.

El nodo *gateway* actúa como puente entre la red de sensores y la estación base. Se encarga de comunicar los elementos que emplean el canal de radio con los elementos que emplean el canal UART. Así, la red de sensores que se comunica mediante el canal de radio puede comunicarse con los elementos que emplean el canal UART, como la estación base.

La estación base es el dispositivo final a donde llegan los mensajes enviados por los nodos sensores. En este caso, la estación base es un PC que se encargará de recibir los mensajes y de almacenar en una base de datos los mensajes que llegan. Por último, mediante un archivo ejecutable en el lenguaje Python el usuario podrá ver los datos que llegan a la estación base a través de la consola y podrá comprobar los datos almacenados en la base de datos.

En la Ilustración 14 se puede ver el esquema básico de la WSN:

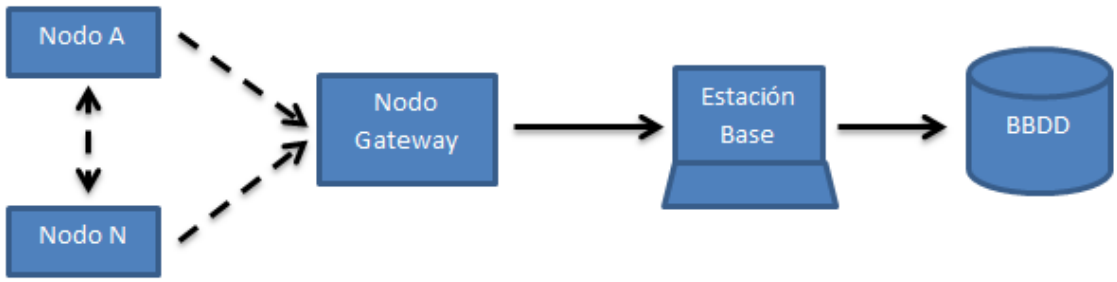

Ilustración 14: Esquema WSN

Antes de ver las partes en detalle debemos hacer un breve repaso sobre el grado de desarrollo que hemos hecho de cada uno de los elementos. La aplicación que se instala en cada uno de los nodos sensores ha sido realizada de forma completa mientras que la aplicación que se instala en el nodo *gateway* ha sido tomada de las aplicaciones a las que se puede acceder una vez se ha instalado TinyOS. El programa de la estación base se ha modificado hasta obtener los elementos deseados. Por último, todos los elementos relacionados con la base de datos han sido implementados desde cero.

4.2.1.1 Nodo sensor

Para que los nodos sensores MicaZ puedan realizar la medición de temperatura necesitan que se les acople una placa de sensores, ya que por defecto no disponen de sensores. La placa que les vamos a acoplar es la MTS300CB. Una vez que el sensor tiene la capacidad para realizar las mediciones y el envío de datos tenemos que darle el código necesario para que pueda realizarlo.

La aplicación tiene que realizar una serie de funciones concretas. Primero, es necesario que el nodo sensor se pueda poner en funcionamiento. Para ello, es necesario que exista una función de arranque. Esto se consigue mediante el componente `MainC`.

Después, el nodo sensor tiene la función de realizar dos mediciones de datos: temperatura y las estadísticas de los protocolos de transporte. Es importante destacar que, en sentido estricto, la lectura de estadísticas no supone una medición sobre un sensor, sino simplemente una toma de los datos almacenados en otros componentes del sistema.

Para poder tomar cada medición necesitamos un intervalo de tiempo para cada una. Así, el tiempo entre mediciones de temperatura se establece en un minuto y la toma de estadísticas se establece en diez minutos.

Así, necesitamos tener un temporizador que se encargue de esta función, lo cual se consigue con el componente `TimerMilliC()`.

Después, pasamos a la medición de temperatura. Para ello necesitamos emplear el componente `TempC()`.

La medición de las estadísticas se realiza mediante los componentes `IPDispatchC` y `UdpC`, con las cuales se toman las medidas de IP, ICMP, ruta y UDP.

Existe la posibilidad de emplear los tres leds que dispone el nodo sensor MicaZ para indicar el cumplimiento de las funciones. Para poder emplear los leds es necesario el componente `LedsC`.

Por último, para el funcionamiento del envío de datos y habilitar la recepción de datos en los nodos sensores es necesario establecer los sockets. Al estar trabajando con UDP emplearemos el componente `UDPSocketC` para acceder a las funciones necesarias.

A continuación, se enumeran los componentes empelados con una breve descripción:

- MainC: es la interfaz que se encarga de la secuencia de arranque. De este modo puede el nodo sensor puede ponerse en funcionamiento.
- LedsC: dispone de las funciones necesarias para poder utilizar los tres leds que tiene el nodo sensor MicaZ.
- TimerMilliC(): este componente ofrece un contador de milisegundos al ser instanciado, lo cual nos permitirá poder tener los dos temporizadores.
- IPDispatchC: este componente ofrece a las aplicaciones instaladas en los nodos sensores recepción IP.
- UDPSocketC(): dispone de las funcionalidades necesarias para poder trabajar con un socket UDP.
- TempC(): este es el componente necesario para poder realizar las mediciones del sensor de temperatura que será acoplado al nodo sensor.
- UdpC: componente donde se incluyen los elementos necesarios para poder obtener los datos del protocolo UDP.

Por último, en la Ilustración 15 se muestra el diagrama de los componentes que se ha obtenido mediante el empleo de nesdoc, una funcionalidad de TinyOS:

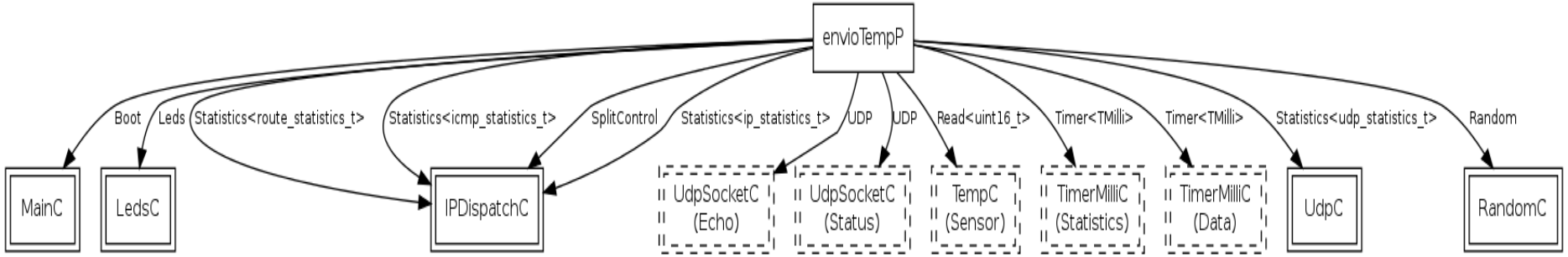

Ilustración 15: Diagrama de componentes

4.2.1.2 Nodo gateway

En el nodo sensor se ha instalado una aplicación para tomar las mediciones y enviarlas a la estación base. El nodo *gateway* se encarga de actuar como un puente entre los nodos sensores y la estación base, por tanto su funcionamiento consiste en recibir los mensajes de la red de nodos sensores y enviarlos a la estación base.

Para ello se ha tomado la aplicación IPBaseStation, que viene con la instalación de TinyOS. Esta aplicación dispone de un sistema de colas para evitar que se pierdan paquetes en el caso de una gran afluencia de tráfico.

En este caso no vamos a entrar en tanto detalle debido a que este código no lo hemos desarrollado y puede encontrarse fácilmente.

4.2.1.3 Estación Base

La estación base en este prototipo se encarga de recibir los datos de la red de sensores y almacenarlos. La recepción de datos se hará mediante la apertura de un socket y la escucha de los datos que van dirigidos a él. Así, solo se recibirán los datos de los nodos sensores de nuestra red, en el hipotético caso de coexistir con otras.

Para poder hacer esto se empleará un programa con código escrito en el lenguaje Python. Este programa tendrá una doble funcionalidad. Primero, se encargará de permitir la recepción de datos de la red y segundo, se encargará de realizar las inserciones en la base de datos de los mensajes que se reciban.

Para la recepción será necesario abrir un socket para que la red de nodos sensores tenga un destino. Después, se comprueba de forma periódica si se han recibido datos para poder tratarlos.

Una vez que tenemos la información se comprobará uno de los campos del mensaje para saber el tipo de mensaje que ha llegado. De este modo se podrá extraer la información y posteriormente se podrá insertar en la tabla correspondiente dentro de la base de datos.

4.2.2 Formato de los mensajes

Existen dos tipos de mensajes, como ya hemos visto. El primero, se encarga de transportar las mediciones de temperatura hasta la estación base. El segundo, obtiene datos sobre los protocolos que toman parte en el transporte de los mensajes.

4.2.2.1 Mensaje de datos

El mensaje de datos tiene cuatro campos, como se puede comprobar en el siguiente fragmento de código que corresponde a la estructura del mensaje:

```
nx_struct data_msg {
  nx_uint8_t tipoMensaje;
  nx_uint16_t sender;
  nx_uint16_t seqno;
  nx_uint16_t datos;
} ;
```

El primer campo, `tipoMensaje`, se emplea para comprobar el tipo de mensaje. Un valor de uno indica que es un mensaje de datos. El segundo campo, `sender`, contendrá el identificador del nodo sensor que emite el mensaje. El campo `seqno` contiene el número de secuencia del mensaje enviado por el nodo sensor. Por último, el campo `datos` contiene la lectura que ha obtenido el nodo sensor a través de la placa de sensores. Este valor se envía en crudo por la red pero será transformado antes de ser introducido en la base de datos para obtener un valor en grados Celsius (grados centígrados).

4.2.2.2 Mensaje de estadísticas

El otro tipo de mensaje contiene los datos sobre las estadísticas de red. Estas estadísticas aportan datos sobre IP, UDP, ICMP y sobre la ruta empleada para la comunicación. De este modo podemos obtener datos sobre las pérdidas de datos cuando se produce la fragmentación del paquete, así como tener datos sobre el estado del envío de paquetes UDP o conocer los costes que tiene la ruta que se emplea en la comunicación.

Este tipo de mensaje dispone de siete campos en lugar de cuatro. A continuación, se muestra el fragmento de código correspondiente:

```
nx_struct udp_report {
  nx_uint8_t tipoMensaje;
  nx_uint16_t seqno;
  nx_uint16_t sender;
  ip_statistics_t    ip;
  udp_statistics_t   udp;
  icmp_statistics_t  icmp;
  route_statistics_t route;
};
```

Los tres primeros campos se mantienen igual que en el mensaje de datos. El campo `tipoMensaje` se emplea para comprobar el tipo de mensaje que ha llegado. Si el valor es dos indica que es un mensaje del tipo estadísticas. El campo `seqno` indica el número de secuencia del mensaje enviado por el nodo sensor. El campo `sender` indica el identificador del nodo sensor que envió el mensaje. Ahora aparecen cuatro campos nuevos, los correspondientes a las estadísticas que van a tomarse. Las estadísticas relativas a IP contienen campos sobre la fragmentación y el sistema de reenvío de paquetes. Dichos campos se incluyen dentro de la estructura `ip_statistics_t`. Las estadísticas relativas a UDP contienen campos con los que se controla el flujo de mensajes UDP y a su emisor. Estos campos se incluyen en `udp_statistics_t`. En el caso de ICMP contiene un campo donde se indican los mensajes de este tipo recibidos, y ese campo se encuentra en la estructura `icmp_statistics_t`. Por último, `route_statistics_t` contiene datos sobre la ruta que se emplea para la comunicación, como son el coste del camino que se toma o la cantidad máxima de saltos que pueden emplearse. Ahora se mostrará para cada estructura una lista con los campos que contiene, de forma que se pueda consultar de forma sencilla:

- ip_statistics_t: `ip.sent`, `ip.forwarded`, `ip.rx_drop`, `ip.tx_drop`, `ip.fw_drop`, `ip.rx_total`, `ip.real_drop`, `ip.hlim_drop`, `ip.sendDone_el`, `ip.fragpool`, `ip.sendinfo`, `ip.sendentry`, `ip.sndqueue`, `ip.encfail` e `ip.heapfree`.
- udp_statistics_t: `udp.total`, `udp.failed`, `udp.seqno`, `udp.sender`.
- icmp_statistics_t: `icmp.rx`.
- route_statistics_t:`route.hop_limit`, `route.parent`, `route.parent_metric`, `route.parent_etx`.

4.2.3 Almacenamiento

A la hora de almacenar los datos que envía la red de nodos sensores necesitamos tener en consideración varios elementos. Primero, tenemos que plantear el método de almacenamiento que vamos a emplear. Para este trabajo se han barajado dos opciones: almacenamiento de datos en una base de datos y volcado de datos en un fichero de texto. A continuación se realizará una breve comparativa entre los dos métodos para poder seleccionar la mejor opción.

A primera vista, queda claro que volcar los datos en fichero de texto es una solución más sencilla que usar la base de datos, ya que solo necesitamos tener un directorio donde guardar el fichero. En cambio, con la base de datos, es necesario generarla, diseñar y crear las tablas, así como un usuario para poder acceder a ellas. Por último, también necesitamos un elemento gestor que nos permita acceder a los datos.

Por otro lado, la base de datos es mucho más cómoda a la hora de analizar los datos que se han obtenido, ya que se pueden recuperar los datos de forma individualizada y realizar consultas concretas, mientras que en el fichero de texto se tienen todos los elementos sin facilidades para trabajar con ellos.

De cara a la implementación de cualquiera de los dos métodos no es necesario incluir nuevos lenguajes de programación. En el caso del volcado de datos a un fichero de texto se puede realizar mediante el lenguaje C. En el de la base de datos, se puede optar por varios lenguajes, pero debido a que vamos a emplear el lenguaje Python para abrir el socket podemos emplear ese mismo archivo para leer los datos y almacenarlo en la base de datos.

Así, el principal motivo de la elección del volcado de datos en un fichero de texto es que es un método sencillo de implementar, ya que el método en sí luego no ofrece ninguna mejora sobre los beneficios que aporta la base de datos. En cambio, la base de datos ofrece los beneficios del manejo de los datos a cambio de una inversión mayor de tiempo para poder crearla de forma adecuada.

4.2.3.1 Diseño

En este trabajo la red de nodos sensores se encarga de enviar dos tipos de mensajes con datos. Uno de esos mensajes contiene los datos de las mediciones de temperatura mientas que el otro contiene las estadísticas de red que se han obtenido.

Para el almacenamiento se empleará una base de datos que siga el modelo relacional, que es el modelo más extendido actualmente. Los elementos en este modelo se almacenan formando relaciones, que son el conjunto de columnas y filas que dan lugar a las tablas, el elemento más característico de una base de datos.

A continuación se muestra una tabla con la cual se entenderá mejor el funcionamiento de la tabla:

Campo 1	Campo 2	…	Campo N
Valor 1,1	Valor 1,2	…	Valor 1,n
…	…	…	…
Valor m,1	Valor m,2	…	Valor m,n

Tabla 51: Ejemplo tabla base de datos

Las columnas están formadas por campos que contienen información sobre un elemento determinado de la tabla. Las filas se conocen por el nombre de tuplas y contienen las relaciones de los campos. El número de atributos que tiene una tabla se conoce como grado mientras que la cantidad de tuplas es la cardinalidad.

Dentro de la tabla existe un campo, o una combinación de campos, que identifica de forma inequívoca cada tupla. Este campo o conjunto de ellos se conoce como clave primaria y solo puede haber una por tabla.

4.2.3.2 Diseño de la base de datos

Primero, necesitaremos una base de datos exclusiva para las tablas de los mensajes que se reciben desde la red de nodos sensores. Además, será necesario generar un usuario específico, que esté vinculado a la base de datos creada, para poder restringir el acceso a los datos almacenados.

Ahora, pasaremos a las tablas que necesitaremos. Serán tres en total, una para cada tipo de mensaje y otra para almacenar la información relativa al nodo sensor emisor. De este modo, dispondremos de una tabla que tendrá los datos del nodo sensor y dos tablas asociadas con los datos enviados por el nodo sensor. Esta asociación se realizará por medio del campo que contiene el identificador del nodo sensor.

Comenzaremos por la tabla que contiene los datos relativos al nodo sensor. El campo más importante será el identificador del nodo sensor, ya que será el campo en común con las otras dos tablas. Esto hará que se puedan separar las mediciones por el identificador del nodo sensor y así tener una visión más específica de los datos. Los campos relativos a las características del nodo sensor se pueden dividir en función del tipo de variable que se empleará para su almacenamiento. Los elementos modelo, radio y microcontrolador serán almacenados en cambios del tipo `char`. Por otro lado, las tres variables que hacen relación a la capacidad de las distintas memorias (flash, EEPROM y RAM) serán almacenadas en variables del tipo `integer`.

Nombre del campo	Tipo	Descripción
Id_nodo_sensor	INTEGER	Contiene el identificador del nodo sensor que emite los datos.
Modelo	CHAR	El modelo de nodo sensor del nodo sensor emisor de los datos.
Radio	CHAR	El modelo de transmisor que tiene el nodo emisor de los datos.
Micro_controlador	CHAR	El tipo de micro controlador que tiene el nodo emisor de los datos.
Memoria_flash	INTEGER	La cantidad de memoria flash que tiene el nodo emisor de datos.
Memoria_EEPROM	INTEGER	La cantidad de memoria EEPROM que tiene el nodo emisor de datos.
Memeoria_RAM	INTEGER	La cantidad de memoria RAM que tiene el nodo emisor de los datos.

Tabla 52: Elementos tabla datos_nodo_sensor

A la hora de diseñar las tablas que contienen los datos enviados por el nodo sensor tenemos que decidir qué datos queremos guardar en cada caso. En estas tablas se ha introducido un campo que indica el momento en el cual se realiza la inserción, `time_recv_gwtimestamp`, mediante la fecha y la hora. Este campo es muy útil ya que nos permite realizar un seguimiento de los momentos de la toma de mediciones, gracias a que la inserción en la base de datos se realiza instantes después, y además nos proporciona un campo idóneo para incluirlo en la clave primaria de la tabla, la cual se verá un poco más adelante en este mismo punto.

En la tabla de los datos de las mediciones de temperatura queremos almacenar los siguientes campos: identidad del nodo sensor, el contador de mensajes y el dato de la temperatura que se obtiene de la placa de sensores. Para estos elementos se utilizarán variables del tipo `integer` a la hora de almacenarlos. Ahora, vamos a considerar el dato que se obtiene de la medición del sensor. Las unidades de este valor no se encuentran en una medida válida, por tanto lo que haremos antes de almacenar los datos será realizar una conversión de ese valor a una unidad de temperatura válida, como son los grados centígrados. Una vez realizamos la conversión también será almacenado en la tabla. Este último valor, la temperatura convertida, será almacenado como una variable de tipo `float`.

Nombre del campo	Tipo	Descripción
Tiempo_insercion	TIMESTAMP	Guarda el momento, mediante el formato de fecha y hora, en el que se realiza la inserción de los datos en la base de datos.
Id_nodo_sensor	INTEGER	Contiene el identificador del nodo sensor que emite los datos.
Contador	INTEGER	Almacena la cantidad de mensajes enviados por el nodo sensor emisor de los datos.
Dato_temeperatura	INTEGER	Valor obtenido por el nodo sensor al realizar la medición de la temperatura con la placa de

		sensores.
temperaturaC	FLOAT	Valor de temperatura obtenido por el nodo sensor convertido a grados centígrados.

Tabla 53: Elementos tabla datos_temperatura

En el caso de la tabla con los valores sobre las estadísticas, se almacenará primero el valor del identificador del nodo sensor emisor. Además, se almacenaran los campos que tienen las diferentes estructuras que se han visto en el punto 4.2.2.2. Así, la tabla contendrá los siguientes campos: `id_nodo_senor`, `ip.sent`, `ip.forwarded`, `ip.rx_drop`, `ip.tx_drop`, `ip.fw_drop`, `ip.rx_total`, `ip.real_drop`, `ip.hlim_drop`, `ip.sendDone_el`, `ip.fragpool`, `ip.sendinfo`, `ip.sendentry`, `ip.sndqueue`, `ip.encfail` e `ip.heapfree`, `udp.total`, `udp.failed`, `udp.seqno`, `udp.sender`, `icmp.rx`, `route.hop_limit`, `route.parent`, `route.parent_metric`, `route.parent_etx`. Todos los elementos de esta tabla se almacenaran en variables de tipo`integer` excepto la toma del tiempo de inserción, que usa el tipo `timestamp`. Por último, cuando en la tabla se haga referencia a L2 se refiere a la capa de enlace de datos.

Nombre del campo	Tipo	Descripción
Tiempo_insercion	TIMESTAMP	Guarda el momento, mediante el formato de fecha y hora, en el que se realiza la inserción de los datos en la base de datos.
Id_nodo_sensor	INTEGER	Contiene el identificador del nodo sensor que emite los datos.
Ip_sent	INTEGER	Total de datagramas IP enviados.
Ip_forwarded	INTEGER	Total de datagramas IP reenviados.
Ip_rx_drop	INTEGER	Cantidad de fragmentos L2 perdidos debido a fallos de 6LoWPAN.
Ip_tx_drop	INTEGER	Cantidad de fragmentos L2 perdidos debido a fallos de enlace.
Ip_fw_drop	INTEGER	Cantidad de fragmentos L2 perdidos durante el reenvío debido a la cola.
Ip_rx_total	INTEGER	Cantidad total de fragmentos L2 recibidos.
Ip_real_drop	INTEGER	Cantidad de fragmentos perdidos.
Ip_hlim_drop	INTEGER	Cantidad de fragmentos perdidos debido al límite de salto.
Ip_senddone_el	INTEGER	Contiene información sobre la finalización del envío.
Ip_fragpool	INTEGER	Cantidad de fragmentos libres en espera.
Ip_sendinfo	INTEGER	Cantidad de estructuras sendinfo disponibles.
Ip_sendentry	INTEGER	Cantidad de entradas de envío disponibles.
Ip_sendqueue	INTEGER	Cantidad de entradas de la cola de envío disponibles.
Ip_encfail	INTEGER	Fragmentos perdidos debido a la cola de envíos.
Ip_heapfree	INTEGER	Espacio disponible en el heap.
Udp_total	INTEGER	Datagramas UDP enviados totales.
Udp_failed	INTEGER	Datagramas UDP enviados fallidos.

Udp_seqno	INTEGER	Número de secuencia del datagrama UDP enviado.
Udp_sender	INTEGER	Dirección del emisor del datagrama UDP.
Icmp_rx	INTEGER	Paquetes ICMP recibidos.
Route_hop_limit	INTEGER	Cantidad máxima de saltos dentro de la ruta.
Route_parent	INTEGER	Indica la ruta padre.
Route_parent_metric	INTEGER	Coste aditivo del camino de la entrada superior en la tabla de enrutamiento.
Route_parent_etx	INTEGER	Coste aditivo de la entrada que actualmente está siendo usada como ruta primaria por defecto.

Tabla 54: Elementos tabla datos_estadisticas

Ahora solo nos queda hablar de la clave primaria de las tablas que vamos a crear. La clave primaria, como ya se ha visto, se encarga de identificar de forma única cada tupla de una tabla. En el caso de la tabla `datos_nodo_sensor`, se ha elegido como clave primaria el campo `id_nodo_sensor`, ya que al existir una sola tupla por cada identificador de nodo sensor podemos identificar las tuplas de forma única. En el caso de las otras dos tablas el identificador del nodo no es un campo único ya que tendremos varias mediciones del mismo sensor, por tanto ese campo ya no podrá ser utilizado como clave primaria. Debido a esto, esta clave tendrá dos campos, el identificador del nodo sensor unido a la medición del tiempo en el que se hace la inserción en la base de datos que por definición es único. Ese nuevo valor contiene la fecha y la hora, por tanto ahora sí se podría realizar la identificación única de tuplas.

Por último, en la Ilustración 16 se puede ver el modelo relacional de la base de datos:

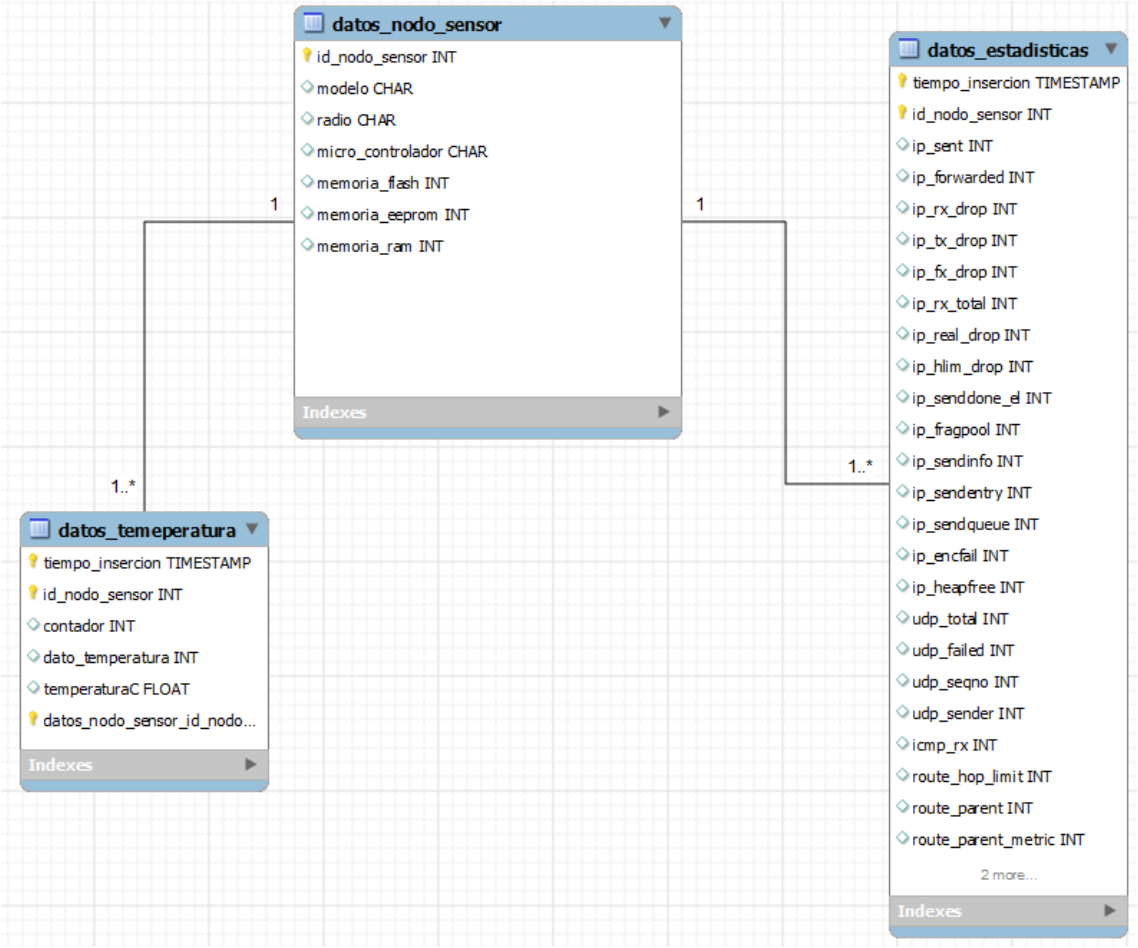

Ilustración 16: Modelo relacional de la base de datos

5. Construcción

En este apartado se verá la implementación que se ha realizado de los diferentes elementos que forman el prototipo. Primero se verá la base de datos, donde se mostrará lo que hace cada script. Después, se explicarán las funcionalidades de cada parte del código de la red de sensores de forma detallada, para comprender cómo funcionan.

Para finalizar la introducción, hay que tener claro que el código al completo se encontrará en los Anexos B, C y D. En este capítulo solo se encontrarán pequeños fragmentos que sean necesarios para la explicación de cada funcionalidad.

5.1Base de datos

En este apartado vamos a ver de forma sencilla como se ha preparado el entorno para trabajar con la base de datos y después pasaremos a explicar de forma detallada los dos scripts que nos permitirán obtener una base de datos operativa. El primero, `bbdd.sql`se encarga de crear la base de datos y el usuario con permisos. Por otro lado, `tablas.sql`se encarga de crear las tablas necesarias, como veremos a continuación.

5.1.1 Entorno

Para la base de datos se ha elegido MySQL, versión 5.5.32. Una vez instalada la versión `mysql-server`, durante la cual se nos pedirá que introduzcamos una contraseña para la cuenta `root`, la cual tiene todos los permisos.

Una vez hecho esto, podemos acceder a través de la consola mediante el comando `´mysql -u root -p´` e introduciendo a continuación la contraseña que hayamos empleado durante la instalación. La opción `-u` se utiliza para introducir un nombre de cuenta a continuación, en este caso `root`. La opción `-p` se utiliza para que nos pida la contraseña. Una vez la hemos introducido podemos realizar diversas acciones, como comprobar las bases de datos existentes o el contenido de éstas.

Para introducir el código que contiene un archivo .sql directamente, debemos emplear el siguiente comando: `´source /<ruta del archivo>´`.

5.1.2BBDD.SQL

En este script se creará el usuario y la contraseña que se empleará para acceder a los datos. Estos datos se pueden modificar a gusto del usuario que vaya a utilizarlo. Como se vio antes, primero eliminamos el usuario que vamos a crear, para evitar posibles conflictos, y después lo creamos. Si este código se ejecuta sobre una base de datos vacía mostrará un error indicando que no existe usuario que borrar, pero la ejecución del código se realizará sin problemas.

```
DROP USER 'dbu';
CREATE USER dbu IDENTIFIED BY 'dbu';
```

Después, se eliminará la base de datos antes de crearla, para evitar conflictos. Una vez hecho esto, se creará la base de datos en la cual se introducirán las tablas y se asignarán permisos al usuario antes creado para que pueda realizar todas las acciones necesarias sobre esta base de datos.

```
DROP DATABASE ddbb;
CREATE DATABASE ddbb;
GRANT ALL ON ddbb.* TO dbu IDENTIFIED BY 'dbu';
```

5.1.3TABLES.SQL

Este script se encarga de la creación de las tablas descritas en el capítulo anterior.

La creación de las tablas se divide en dos partes. Primero, se tiene que seleccionar la base de datos que se va a usar, ya que en caso contrario no se sabría donde se han creado las futuras tablas. Ahora, una vez que ya tenemos seleccionada la base de datos donde se van a crear las tablas tenemos que borrar las tablas que puedan existir con el mismo nombre que las nuevas. Primero, se vacía de datos la tabla y después se borra la propia tabla. Una vez hecho esto, solo hay que crear la nueva tabla con los campos que se necesiten.

```
USE ddbb;
DELETE FROM datos_temperatura;
DROP TABLE IF EXISTS datos_temperatura CASCADE;
CREATE TABLE datos_temperatura (
    tiempo_insercion timestamp,
    id_nodo_sensor int,
    contador int,
    dato_temperatura int,
    temperaturaC float,
    PRIMARY KEY (tiempo_insercion, id_nodo_sensor)
);
```

En el fragmento de código anterior podemos apreciar los diferentes campos de la tabla datos_temperatura, así como su clave primaria. Las otras tablas mantienen el mismo formato, solo varían los campos que incluye la tabla y los campos que forman la clave primaria.

5.2 Creación del prototipo

Como ya se ha visto, el prototipo consta de tres partes bien diferenciadas: nodos sensores, nodo *gateway* y estación base. Así, se mostrarán por separado cada uno de ellos para poder ver en profundidad cómo funcionan.

5.2.1 Aplicación en los nodos sensores

La aplicación en los nodos sensores realiza varias funciones. Primero, debe ser capaz de tomar la medición de temperatura de la placa de sensores que se ha unido al nodo sensor MicaZ. Además, debe ser capaz de realizar estas mediciones en una frecuencia determinada, establecida mediante un temporizador. Similarmente, para obtener las estadísticas de red se usará una frecuencia más baja.

A continuación se verán las funcionalidades de forma más detallada:

5.2.1.1Temporizador

Una vez que el sensor se pone en marcha se activan dos temporizadores, uno por cada tipo de mensaje: datos y estadísticas. Esto se hace mediante el siguiente comando:

```
call StatusTimerS.startPeriodic(REPORT_PERIOD_S);
call StatusTimerD.startPeriodic(REPORT_PERIOD_D);
```

Las llamadas se diferencian entre sí mediante las letras S y D de estadísticas y datos.

Las variables`REPORT_PERIOD_S` y `REPORT_PERIOD_D`tienen valores diferentes, ya que el tiempo entre mediciones no es igual para las dos funciones.

Una vez que el temporizador expira se genera un evento asociado y se implementa el manejador asociado a cada uno de los eventos:

```
event void StatusTimerD.fired()
event void StatusTimerS.fired()
```

Dentro de esta función se asignarán valores a los campos del mensaje y se llamará a la función de envío, lo cual se verá más adelante en profundidad.

5.2.1.2 Envío

El envío de datos se compone de tres partes: los datos del destinatario, los datos a enviar y la función de envío.

Los datos del destinatario se almacenan en una estructura definida como `struct sockaddr_in6 route_dest`, que contiene dos campos: `uint16_t sin6_port` y `struct in6_addr sin6_addr`, donde se guardan de forma respectiva el puerto y la dirección de destino.

Mediante el siguiente fragmento de código se explicará cómo se establecen estos datos:

```
#ifdef REPORT_DEST
    route_dest.sin6_port = hton16(6500);
    inet_pton6(REPORT_DEST, &route_dest.sin6_addr);
#endif
```

La variable `REPORT_DEST` se encuentra en el archivo Makefile y consiste en la dirección IPv6 de destino, como se puede ver a continuación.

```
CFLAGS += -DREPORT_DEST=\"2001:470:1f12:e4::2\"
```

Esta dirección debe ser configurada de forma manual previamente a la instalación de la aplicación en el nodo sensor. De este modo, todos los nodos sensores que tengan la variable configurada con una dirección IPv6 enviarán sus datos a esa dirección.

Por otro lado, el puerto de destino se establece mediante la segunda línea del fragmento de código anterior. El valor del puerto se introduce también de forma manual.

Por último, la línea que contiene `inet_pton6 (REPORT_DEST, &route_dest.sin6_addr)` se encarga de convertir los datos de la forma de presentación, usualmente ASCII, al formato de red.

Los datos que se envían variarán en función del tipo de mensaje. Dentro de `StatusTimerX.fired()` se guardarán en las variables correspondientes los datos que se deban enviar excepto en el caso de la medición de temperatura, que se almacenara en la función que realiza esa acción. En la sección de los tipos de mensaje se verán en profundidad los diferentes campos.

Por último, solo nos falta la función de envío:

```
call Status.sendto(&route_dest, &datos, sizeof(datos));
call Status.sendto(&route_dest, &stats, sizeof(stats));
```

En el primer campo se coloca el destinatario, en el segundo los datos que se quieren enviar y en el tercero el tamaño que tienen los datos a enviar.

Con esto podemos concluir con los elementos encargados del envío de los mensajes.

5.2.1.3 Recepción

Los nodos sensores pueden recibir datos de otros nodos sensores o de la estación base. Debido a esto, se establece un socket de recepción de la siguiente manera:

```
call Echo.bind(7);
call Status.bind(6501);
```

En la aplicación, se abren dos sockets del tipo UDP. El primer socket, Status, será utilizado para enviar datos desde el nodo sensor a la estación base. El segundo socket, Echo, estará a la espera de recibir posibles datos.

La función *`bind(puerto)`* se encarga de asociar el puerto que se introduzca al socket que estemos usando. En este caso, se encarga de asociar el puerto 7 al socket que emplearemos para recibir datos y de asociar el puerto 6501 al socket que emplearemos para enviar los datos.

Después, está la función `Echo.recvfrom` que se encarga de recibir el mensaje y reenviarlo mediante una función de envío como la vista en la sección anterior.

```
  event void Echo.recvfrom(struct sockaddr_in6 *from, void *data,
                           uint16_t len, struct ip_metadata *meta) {

  call Echo.sendto(from, data, len);
}
```

Esto se hace así debido a que el nodo sensor no realiza ninguna función con los mensajes recibidos y si no los reenviara serían mensajes perdidos.

5.2.1.4 Medición

Primero, hay que tener claro qué sensores de la placa van a usarse, ya que en función de lo que se quiera medir se emplearán unos sensores u otros. Así, con la placa de sensores que se ha empleado en este prototipo podemos obtener mediciones de temperatura, acústicas y luminosas.

Nosotros solo utilizaremos el medidor de temperatura, por tanto, solo tendremos un función que se encargue de la lectura de los sensores. Esta función sencillamente se encargará de guardar en una variable, que será incluida en el mensaje que se envíe, la medición obtenida por el sensor.

```
event void Read.readDone(error_t result, uint16_t data) {
  if (result == SUCCESS){
    datos.datos = data;
  }
 }
```

En el fragmento de código superior se puede ver claramente la explicación anterior. Para que esto funcione, solo hace falta que en el código que se encarga de tratar con las interfaces se incluyan las siguientes:

```
components new TempC() as Sensor;
UDPEchoP.Read -> Sensor;
```

Una vez que tenemos todo esto solo nos falta realizar la llamada a la función. Esto se realizará en la parte del código correspondiente a la finalización del temporizador, `StatusTimerX.fired()`. Con el comando `` `callRead.read();` `` llamamos a la función que toma el valor de la temperatura y esperamos la notificación del evento asociado `Read.readDone()`.

5.2.2 Nodo Gateway

En este apartado se utiliza una aplicación diferente a la de los nodos sensores debido a que cumple una funcionalidad muy diferente. Este elemento funciona como puente entre la red de nodos sensores y la estación base.

Por esto, se instalará en el nodo *gateway* la aplicación `IPBaseStation`, la cual viene como aplicación en la instalación de TinyOS.

Como ya hemos visto antes, al funcionar como un puente entre la red y la estación base las principales funciones que realiza son la recepción y el envío de mensajes de una zona a la otra. Dispone además de un sistema de colas para evitar que los menajes se pierdan en caso de tener una cantidad de recepciones elevadas en muy poco tiempo.

Este elemento no fue necesario modificarlo debido a que se ajustaba perfectamente a las necesidades que teníamos.

5.2.3 Estación base

En este punto se verán dos funciones: la recepción de los datos enviados por los nodos sensores y el almacenamiento de los datos en la base de datos. Para la implementación de ambas funciones se ha realizado un programa escrito en Python, cuyos detalles de implementación se describen a continuación.

5.2.3.1 Recepción de datos

Para esta función se emplean pocas líneas puesto que lo único que se necesita es acceder al contenido del mensaje que viene desde el nodo sensor emisor de datos.

```
1   s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
2   s.bind(('', port))
3
4    while True:
5
6        data, addr = s.recvfrom(1024)
7
8        if (len(data) > 0):
9
10           rpt = UdpReport.UdpReport(data=data, data_length=len(data))
```

Con las dos primeras líneas, se abre el socket para la recepción de datos. La primera línea indica las cualidades del socket en cuanto al tipo de dirección; AF_INET6 indica que trabaja con IPv6 y SOCK_DGRAM indica que es un socket del tipo UDP. Con la segunda línea se realiza el `bind` del socket para asociar el socket al puerto de la estación base.

En la cuarta línea se emplea un bucle `while` para que la lectura de datos se realice de forma indefinida mientras el archivo Python esté ejecutándose.

En la sexta línea se obtienen todos los datos del mensaje que se ha recibido, de tal manera que después se pueda trabajar con ellos.

Con la octava línea se comprueba que el mensaje tenga contenido. Si trae datos se pasa a la siguiente línea, donde se llama al fichero asociado `UDPReport`.py para que los datos por pantalla se muestren de acuerdo a cómo se indica en `UDPReport`.py.

Este fichero, `UDPReport`.py, se encarga de, una vez obtenido el contenido del paquete, mostrar por pantalla los diferentes campos así como sus valores.

5.2.3.2Almacenamiento

A continuación comentaremos cómo se realiza la inserción de datos en las tablas creadas con los scripts anteriores.

Para hacerlo, modificaremos el archivo Python`Listener`.py que se encarga de abrir el socket y leer los datos que son recibidos por la estación base.

```
conn = MySQLdb.connect (host = "localhost",
                        user = "dbu",
                        passwd = "dbu",
                        db = "ddbb")
cursor = conn.cursor()
```

Lo primero que haremos será establecer una conexión con la base de datos con el usuario y la contraseña que se crearon en `BBDD.sql`. Una vez realizada la conexión se establecerá un objeto de tipo cursor, última línea del fragmento de código anterior, para poder ejecutar futuras acciones, como la inserción de datos.

```
data, addr = s.recvfrom(1024)
datosSensor = list(data)
```

Después, una vez leídos los datos se guardan en una variable, en este caso la variable `data`. En la segunda línea se emplea la función `list`, que se encargará de dividir el contenido del campo `data` en una estructura similar a un `array`.

A continuación, se comprueba el campo encargado de diferenciar los mensajes de datos de los mensajes de estadísticas para poder acceder al apartado adecuado para el tratamiento de los datos.

```
if(ord(datosSensor[0]) == 1):
```

El mensaje de datos contiene tres campos: identificador del nodo sensor, contador y dato temperatura. Al ser campos que ocupan dos bytes, es necesario unirlos antes, lo cual se hace de la siguiente manera:

```
contador = ord(datosSensor[3]) << 8 | ord(datosSensor[4])
```

Ahora, para el campo `datos`, se realiza una serie de operaciones para convertir el valor leído por el sensor en un valor en grados Celsius se empleará la siguiente fórmula:

$$1/T(K) = a + b \times \ln(R_{thr}) + c \times [\ln(R_{thr})]^3$$

El contenido de las diferentes variables que se emplean se puede encontrar a continuación:

$R_{thr} = R1(ADC\_FS-ADC)/ADC$
$a = 0.00130705$
$b = 0.000214381$
$c = 0.000000093$
$R1 = 10\ k\Omega$
$ADC\_FS = 1023$
ADC = output value from Mote's ADC measurement.

El valor ADC se refiere al dato obtenido por el sensor que se encuentra en el nodo sensor.

Por último, se insertan los datos en la tabla con el comando:

```
cursor.execute('''INSERT INTO datos_sensor
(id_nodo_sensor,contador,dato_temperatura, temperaturaC)
values (%s,%s,%s,%s)''',(idN,contador,dato, tempC))
```

En la sentencia de inserción hay cuatro campos y no tres, lo cual se debe a que tras haber obtenido el valor en grados Celsius se insertan los dos valores.

Si por el contrario el valor del mensaje corresponde al de estadísticas, hay que tener en cuenta que algunos campos ocupan dos bytes y otros solo uno. Una vez unidos los campos que ocupan dos bytes en un solo campo, procedemos a llamar a la llamada que inserta los valores. En este caso no se muestra al ser como la anterior solo que usando en vez de cuatro inserciones son veinticinco.

5.3 Líneas de código

Para poder tener una visión real del trabajo realizado se va a incluir a continuación una tabla resumen donde se muestren los archivos de código desarrollados así como las líneas de código implementadas en cada uno de ellos.

La tabla tendrá tres apartados: zona del prototipo, nombre del programa y líneas de código. La zona del prototipo servirá para saber en qué parte funciona ese código y podrá tener los valores: nodo sensor, nodo *gateway* y estación base. El nombre del programa indicará el elemento en cuestión que se está analizando y, por último, las líneas de código mostrarán la cantidad de líneas que se han generado. Esto se debe a que en algunos elementos no se han realizado modificaciones ya que cumplen exactamente con nuestras necesidades, como es el caso de la aplicación que se encuentra en el nodo *gateway.*

Zona del prototipo	Nombre del programa	Líneas de código
Nodo sensor	envioTemp	224
Nodo *gateway*	IPBaseStation	0
Estación base	Listener.py	102
Estación base	Base de datos	63

Tabla 55: Trabajo realizado

6. Evaluación

En este capítulo se presentan las pruebas que se han realizado para validar el prototipo desarrollado. Primero, se verá cómo funciona el prototipo y los elementos que se deben emplear para su correcto funcionamiento. Después, se mostrarán los diferentes escenarios que se han empelado para la validación del prototipo.

6.1 Funcionamiento del prototipo

A la hora de poner el prototipo en funcionamiento es necesario ver los elementos que se van a emplear. Primero, necesitaremos los elementos físicos. En este caso, se han empleado dos nodos sensores modelo MicaZ con sus correspondientes placas de sensores modelo MTS300, un nodo *gateway* modelo MIB520 al que se ha adjuntado otro nodo sensor y conectado a la estación base mediante un cable USB. Por último, como estación base se necesita un PC o portátil.

El Anexo A describe la instalación de TinyOS 2.1.1 sobre el PC.

6.1.1 Nodos Sensores

Para poder instalar una aplicación sobre un nodo sensor se utiliza el nodo *gateway,* el cual actúa como placa programadora del nodo proporcionando conectividad con el PC. Para poder instalar la aplicación necesitamos colocarnos dentro del directorio que contiene la aplicación. Para llegar a ella vamos a usar la variable de entorno `$TOSROOT`, que nos indicará la ruta donde se encuentra la carpeta tinyos-2.1.1 gracias a las variables globales que se configuraron al instalar TinyOS. Por otro lado, la aplicación se colocó dentro de la carpeta `apps` de TinyOS. Por tanto, emplearemos el siguiente comando para acceder al lugar:

```
cd /$TOSROOT/apps/envioTemp
```

Una vez en la carpeta envioTemp utilizamos otro comando para instalar la aplicación en el nodo sensor:

```
make micaz blip install.X mib520,/dev/ttyUSBY
```

Este comando dispone de varias partes. Los elementos encargados de que la aplicación se instale son `make` e `install.X`. La X se cambiará por un número que indicará el identificador de ese nodo sensor. En las pruebas hemos empleado el número tres y el número cuatro para los dos nodos sensores. El elemento `micaz` se emplea para indicar el modelo del nodo sensor. Si en vez de MicaZ estuviéramos empleando nodos sensores TelosB se incluiría `telosb` en lugar de `micaz`. La opción `blip` se incluye debido a que estamos empleando 6LoWPAN y así puede incluir algunos archivos que de forma normal no utiliza. Con `mib520` se indica el elemento que se emplea para la conexión con el PC, es decir, el modelo de nodo *gateway*. Por último, `/dev/ttyUSBY` indica la ruta hacia la conexión USB y la Y se utilizará para indicar que conexión física se está empleando.

La aplicación dispone de un documento Makefile donde se establecen ciertas instrucciones que se deben seguir a la hora de empelar el comando make.

```
1  SENSORBOARD=mts300
2  COMPONENT=envTemp
3
4  CFLAGS += -DCC2420_DEF_CHANNEL=15
5  CFLAGS += -DREPORT_DEST=\"2001:470:1f12:e4::2\"
6
7  include $(MAKERULES)
```

En la línea uno se establece el modelo de placa de sensores que se va a emplear. La segunda línea indica el componente de máximo nivel de la aplicación. En la línea cuatro se establece el canal que se empleará para la comunicación. En la línea cinco se establece la dirección de destino a la cual se enviarán los datos obtenidos en el nodo sensor. Por último, en la línea siete se incluyen las reglas para la recompilación.

6.1.2 Nodo gateway

Una vez tenemos instalado la aplicación en el nodo sensor pasaremos a instalar la aplicación dentro del nodo *gateway*. Para ello haremos lo mismo que en el caso de los nodos sensores, primero iremos a la ubicación de la aplicación y después se instalará.

Los comandos empleados serán:

```
cd /$TOSROOT/apps/IPBaseStation/
```

```
make micaz blip install mib520,/dev/ttyUSB0
```

Las diferencias entre estos comandos y los anteriores son el nombre de la aplicación que se va a instalar y que en este caso al nodo *gateway* no hay que darle un identificador.

6.1.3 Estación base

Ahora, vamos a pasar a los elementos que deben ejecutarse en la estación base. Se verán tres elementos: Router Advertisement Daemon, Driver y la puesta en marcha de Listener.py.

6.1.3.1 Router Advertisement Daemon

Es un programa que actúa en segundo plano, por ser un demonio, y se encarga de asignar y configurar automáticamente direcciones IPv6 dentro de una red, en este caso una PAN.

Es posible que este programa no se encuentre instalado, pero esto se soluciona mediante:

```
sudo apt-get install radvd
```

Después, solo habrá que configurar un fichero, `radvd.conf` que se encontrará en `/etc/`. En caso de no existir habrá que crearlo.

Dentro, incluiremos lo siguiente:

```
interface eth0 {
    AdvSendAdvert on;
    prefix 2001:470:1f12:e4::/64{
    };
};
```

El código se establece para la interfaz eth0. La primera opción que hemos activado, y la única, es `AdvSendAdvert`. Con esto conseguimos habilitar los anuncios y solicitudes de enrutamiento. Después, con `prefix`, establecemos el prefijo de la subred para todos los nodos que forman parte de ella.

Una vez configurado, el demonio se arranca mediante el comando `sudo radvd`.

6.1.3.2 Driver

Este elemento se instalará en la estación base y servirá para controlar el prefijo IPv6 que se usa en la subred que vamos a emplear, así como de seleccionar el canal 802.15.4 que usará la red. Por último, puede establecer un mecanismo de descubrimiento de vecinos en otro interfaz para que máquinas de la red puedan descubrir nuevos hosts.

Para ello, solo tenemos que realizar la instalación ya que los elementos necesarios se obtienen al instalar TinyOS 2.1.1. Para la instalación hemos de seguir los siguientes pasos:

El Driver tiene una dependencia con la librería `libmote.a`por lo que deberá ser previamente instalada. Para ello, vamos a `$TOSROOT/support/sdk/c/sf` y ejecutamos los siguientes comandos:

```
>./bootstrap

> ./configure

>make
```

De este modo tendremos la librería construida e instalada. Ahora, nos movemos a `$TOSROOT/support/sdk/c/blip` y ejecutamos los siguientes comandos:

```
>./bootstrap.sh

>./configure

>make
```

Ahora solo falta configurarlo. Lo haremos mediante el archivo `serial_tun.conf`. Dentro de este archivo vamos a modificar tres variables. Primero, la dirección que tendrá la estación base. En nuestro caso es `2001:470:1f12:e4::2`. Después, como se vio antes, se puede asignar otra interfaz para el descubrimiento de vecinos. En este caso, asignamos la interfaz de auto envío `lo`. Por último, el canal del estándar de comunicación 802.15.4. Las opciones válidas van desde el canal 11 al canal 26. Nosotros elegimos el 15 pero se puede escoger cualquier canal.

Con esto hemos concluido la instalación. Para ponerlo en funcionamiento, desde la ubicación actual utilizaremos el siguiente comando:

```
sudo driver/ip-driver /dev/ttyUSBX micaz
```

6.1.3.3 Listener.py

Este programa es el encargado de recibir los datos que envían los nodos sensores y de insertar esos datos dentro de la base de datos. Para que funcione solo tenemos que ejecutar el archivo mediante el comando `python Listener.py`.

6.2 Pruebas

En este apartado se plantearán los distintos escenarios establecidos así como las pruebas que se han realizado para validar el funcionamiento del prototipo de red.

Como se vio en el apartado anterior, las pruebas se han realizado con una red de sensores inalámbricos compuesto por los siguientes elementos:

- Dos nodos sensores.
- Un nodo *gateway*.
- Una estación base.

Los componentes de la red variarán en función del tipo de prueba que se realice. Antes de explicar en profundidad cada prueba se especificarán los componentes de la red mediante un diagrama explicativo.

Las pruebas serán progresivas, comenzarán siendo acciones sencillas, como puede ser el envío de datos, y a medida que avanzan serán acciones más complejas hasta aunar todas las funcionalidades en una sola prueba.

Se realizarán con cinco escenarios diferentes, de tal manera que queden cubiertas todas las funciones que puede desempeñar el prototipo. Así, en cada escenario se realizarán un número variable de pruebas. Esto se debe a que algunas pruebas se validan mejor dividiéndolas en funcionalidades menores.

La siguiente imagen muestra el formato básico de la red de sensores. El Nodo A se encarga de tomar de forma periódica los valores temperatura y las estadísticas y las envían a la estación base a través del Nodo Gateway. Además, hay que configurar en el Nodo A la dirección de destino, que será la de la estación base, y el puerto de destino.

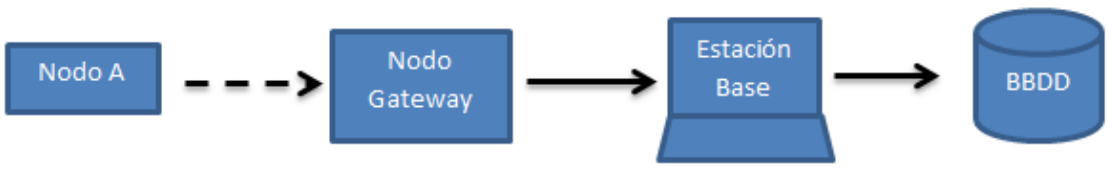

Ilustración 17: Esquema básico para la evaluación

Para validar que los mensajes llegan correctamente y almacenarlos en la base de datos se empleará un fichero con código en Python que recibirá los mensajes que vayan al puerto y a la dirección correcta. Además, se mostrarán por pantalla los datos del mensaje y se almacenarán en la tabla correspondiente de la base de datos. Este fichero será único para los dos tipos de mensaje.

A continuación, se mostrarán los cinco escenarios que se han diseñado para probar las funcionalidades. Los primeros cuatro escenarios usan UDP y el quinto será con TCP. El primer escenario sirve para comprobar el envío de datos y la toma de mediciones. El segundo para validar el formato de envío mediante el envío a direcciones y puertos erróneos. El tercero se centra en el envío de un nodo a otro nodo de la red. El cuarto se centra en el envío a una estación base ajena a la red. El último escenario, el quinto, se centra en repetir los escenarios uno y tres con TCP.

A la hora de entrar en cada prueba esto es lo que se encontrará: primero, el diagrama de la red de sensores para la prueba. Después, se especificarán los elementos que se deberán configurarse en el nodo emisor o nodos emisores y en la estación base. Por último, se comprobará si los datos han llegado a la estación base o no.

6.2.1 Escenario 1

El escenario 1 está diseñado para validar el correcto funcionamiento del envío de mensajes del nodo sensor a la estación base a través del nodo Gateway. Además, se validará la medición de datos.

El envío de mensajes se validará de varias maneras. Primero, de forma sencilla con un envío del mensaje de datos sin realizar la medición. Después se realizará con dos nodos emisores para validar que la estación base es capaz de recibir mensajes de varios nodos diferentes.

La toma de datos se validará mediante la inclusión de ese dato en el mansaje que se enviará a la estación base.

6.1.1.1 Envío de datos simple

El esquema del envío de datos es el siguiente:

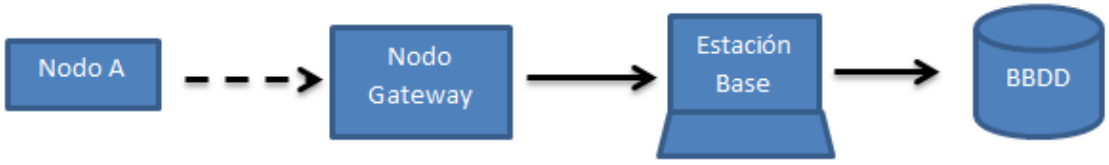

Ilustración 18: Envío de datos simple

En el Nodo A se ha instalado la aplicación envioTemp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

Esta es la configuración que se realiza en el Nodo A.

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::3	2001:470:1f12:e4::2	6500	6500

Tabla 56: configuración envío de datos simple

Una vez hecho esto y con todos los elementos configurados tal y como se ha visto en el punto anterior, ponemos en funcionamiento el sensor para que comience a transmitir los datos. En la pantalla se ve que la estación base está recibiendo mensajes desde la red de nodos sensores, por tanto, el resultado de la prueba es correcto.

6.2.1.2 Envío de datos con dos nodos emisores

Este será el esquema de red:

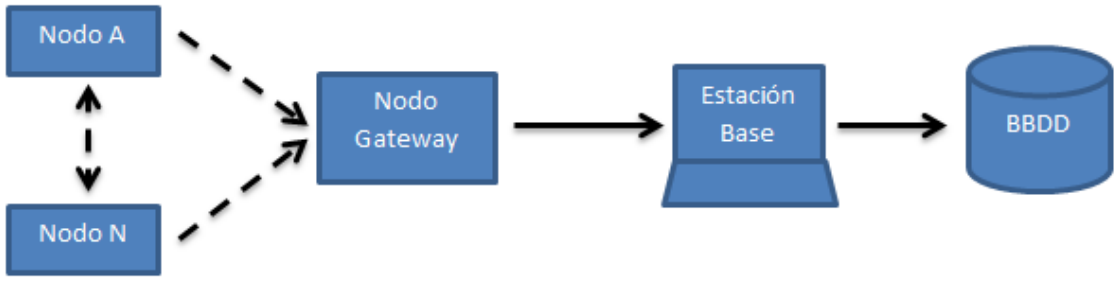

Ilustración 19: Envío de datos con dos nodos emisores

El Nodo A y el Nodo B se ha instalado la aplicación envioTemp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

A continuación se muestra la configuración del Nodo A y del Nodo B:

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::3	2001:470:1f12:e4::2	6500	6500
B	2001:470:1f12:e4::4	2001:470:1f12:e4::2	6500	6500

Tabla 57: Configuración envío de datos con dos nodos emisores

Tras configurarlos correctamente se encienden y vemos que en la consola se pueden ver los mensajes intercalados con los dos identificadores, uno por cada nodo, por tanto el resultado de la prueba es correcto.

6.2.1.3 Medición de datos

El esquema de este punto será el siguiente:

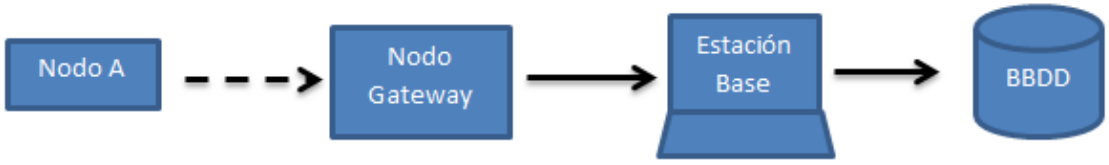

Ilustración 20: Medición de datos

En el Nodo A se ha instalado la aplicación envioTemp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

Esta es la configuración que se realiza en el Nodo A:

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::3	2001:470:1f12:e4::2	7000	7000

Tabla 58: Configuración medición de datos

La diferencia entre esta prueba y la de envío de datos simple, a primera vista idénticas, es que en este caso se incluye una placa de sensores en el Nodo A para poder obtener las mediciones de temperatura. Así, una vez que hemos comprobado que el envío funciona solo tenemos que fijarnos en la consola para comprobar que los datos de temperatura son coherentes con la temperatura en el laboratorio.

Tras comprobar que los mensajes llegan correctamente podemos concluir que el primer escenario, donde se ha evaluado de forma satisfactoria el envío de datos y la toma de los datos de las mediciones de temperatura, funciona correctamente.

6.2.2 Escenario 2

En este escenario se quiere evaluar el funcionamiento del envío de datos con direcciones y puertos erróneos. En este escenario se realizarán dos pruebas: primero, se evaluará el envío a una dirección IPv6 incorrecta con los puertos correctos; en segundo lugar, se hará lo contrario, dirección IPv6 correcta y puertos incorrectos. Como prueba de control tenemos las pruebas del primer escenario donde se vio cómo el envío de datos se realizaba de forma satisfactoria si los datos son correctos.

6.1.2.1 Dirección IPv6 errónea

El modelo de la red es el siguiente:

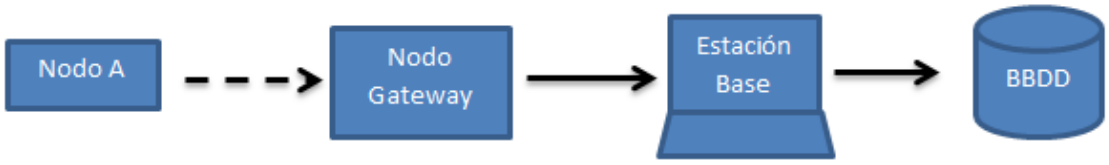

Ilustración 21: Dirección IPv6 errónea

En el Nodo A se ha instalado la aplicación envioTemp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

Esta es la configuración que se realiza en el Nodo A.

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::3	2001:470:1f12:e4::8	6500	6500

Tabla 59: Configuración dirección IPv6 errónea

Ahora, tras encender el Nodo A vemos que en la consola no hay ningún movimiento. De este modo, podemos concluir que si la dirección IPv6 es errónea los mensajes no llegan correctamente.

6.2.2.2 Puertos erróneos

El modelo de la red es el siguiente:

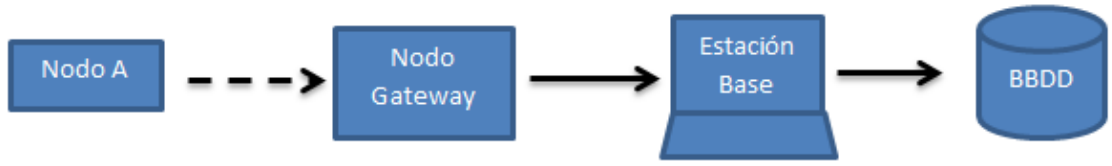

Ilustración 22: Puertos erróneos

En el Nodo A se ha instalado la aplicación envioTemp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

Esta es la configuración que se realiza en el Nodo A.

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::3	2001:470:1f12:e4::2	6500	6000

Tabla 60: Configuración puertos erróneos

Ahora, tras encender el Nodo A vemos que en la consola no se aprecia comunicación debido a que no se están recibiendo mensajes. Por tanto, al igual que antes, podemos concluir que los mensajes no llegan a la estación base si los puertos no están bien configurados.

6.2.3 Escenario 3

En este escenario se valida si un nodo sensor es capaz de enviar datos a otro nodo sensor de la red. Para ello se realiza una donde se establecen los datos de configuración de tal manera que un nodo sensor realiza envíos a la estación base y el segundo nodo hace los envíos al primero. Después, se realiza una segunda prueba donde se muestra cómo si no se establecen los datos de configuración de forma correcta no se puede realizar la comunicación entre nodos.

6.2.3.1 Comunicación entre nodos

Así, el esquema es el siguiente:

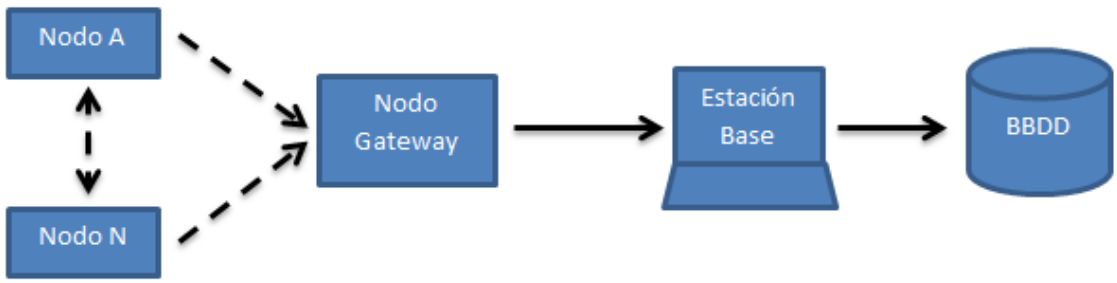

Ilustración 23: Comunicación entre nodos

El Nodo A y el Nodo B se ha instalado la aplicación envioTemp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

A continuación se muestra la configuración del Nodo A y del Nodo B:

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::3	2001:470:1f12:e4::2	6500	6500
B	2001:470:1f12:e4::4	2001:470:1f12:e4::3	7	6500

Tabla 61: Configuración comunicación entre nodos

En el código se ha visto que los nodos sensores realizan la función bind() por defecto en el puerto 7, por tanto la idea que hay tras esta prueba es que el Nodo B envíe los mensajes a la dirección del Nodo A y al puerto que usa dicho nodo. Así, se podrá comprobar si la luz asociada a la recepción de mensajes, la roja, se enciende o no.

Tras encender ambos nodos se puede comprobar que la luz roja del Nodo A se enciende, por tanto se puede concluir que el Nodo A recibe los mensajes del Nodo B y que por tanto la comunicación entre nodos en una red es posible con 6lowpan.

6.2.3.2 Comunicación fallida entre nodos

Se utiliza el siguiente esquema de red:

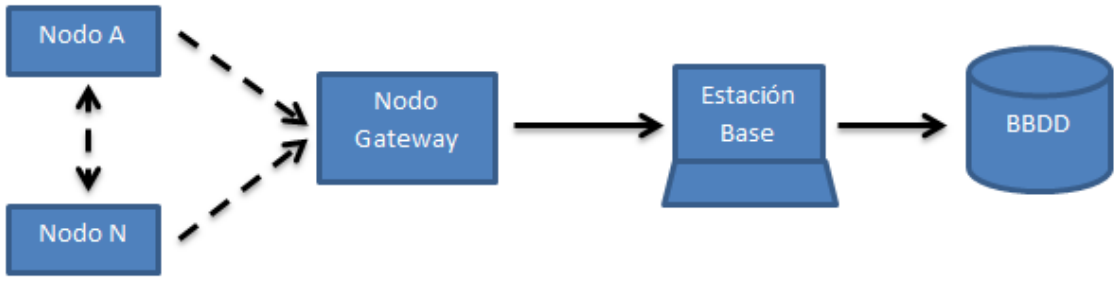

Ilustración 24: Comunicación fallida entre nodos

El Nodo A y el Nodo B se ha instalado la aplicación envioTemp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

A continuación se muestra la configuración del Nodo A y del Nodo B:

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::3	2001:470:1f12:e4::2	6500	6500
B	2001:470:1f12:e4::4	2001:470:1f12:e4::3	6500	6500

Tabla 62: Configuración comunicación fallida entre nodos

En esta segunda prueba se quiere probar si la comunicación entre nodos es efectiva sin establecer como puerto de origen aquel que abre el nodo que va a recibir los datos. Si nos fijamos en los datos de configuración de la tabla, se puede apreciar como la única diferencia

entre ambos es que el puerto de destino del Nodo B no es el mismo, en este caso es el que se abre en la estación base.

Una vez que encendemos los nodos, vemos cómo solo llegan los mensajes del Nodo A a la estación base.

6.2.4 Escenario 4

En este escenario se valida el envío de mensajes desde el nodo sensor a una estación base que está fuera de la red de sensores. Para validarlo, en la estación base de destino se utilizará el programa Wireshark para comprobar si se reciben los mensajes.

El esquema de la red de sensores es el siguiente:

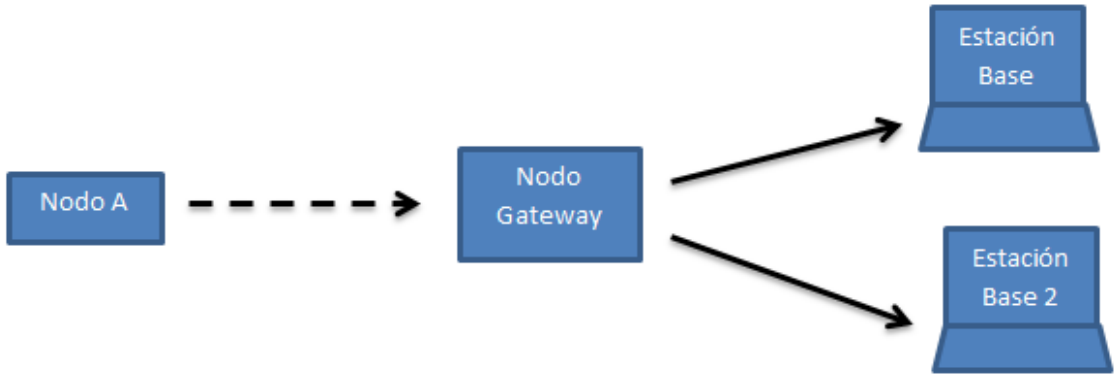

Ilustración 25: Escenario 4

En el Nodo A se ha instalado la aplicación envioTemp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

Esta es la configuración que se realiza en el Nodo A.

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::3	2001:0:5ef5:79fb:3c85:3ce2:3f57:fe9a	6500	6500

Tabla 63: Configuración escenario 4

Al encender el nodo comienza la emisión de datos y debemos fijarnos en el programa Wireshark. Se puede comprobar que no llega ningún mensaje desde el nodo sensor, por tanto podemos decir que no se puede acceder a una estación base que se encuentre fuera de la red.

6.2.5 Escenario 5

En este último escenario, vamos a realizar algunas de las pruebas anteriores utilizando el protocolo TCP en lugar del protocolo UDP.

Primero, se realizarán las pruebas básicas donde se comprueba que TCP funciona mediante tres pruebas. Para finalizar, se realizarán las pruebas del escenario tres con TCP.

Para estas pruebas se ha modificado el código que se empleaba en los nodos sensores en los escenarios anteriores. Ahora, se emplea TCP en lugar de UDP dejando el resto del código intacto.

6.2.5.1 Ping6

El esquema a emplear será el básico:

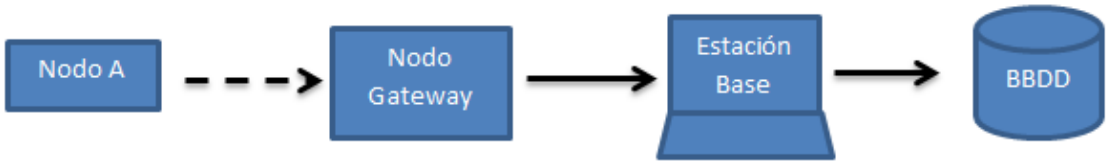

Ilustración 26: Ping6

En el Nodo A se ha instalado la aplicación envioTempTcp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

Esta es la configuración que se realiza en el Nodo A.

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::1	2001:470:1f12:e4::2	6500	6500

Tabla 64: Configuración ping6

Esta prueba es muy sencilla, ya que mediante una consola se intentará obtener respuesta al usar el comando ping6, que consiste en el envío y respuesta de mensajes ICMP, lo cual se emplea para ver si el estado de la red es correcto.

Tras abrir la consola e introducir el comando "ping6 2001:470:1f12:e4::1 y obtenemos una respuesta afirmativa por parte del nodo sensor, por tanto podemos concluir que se puede usar este comando.

6.2.5.2 Nc6 TCP

El esquema a emplear será el básico:

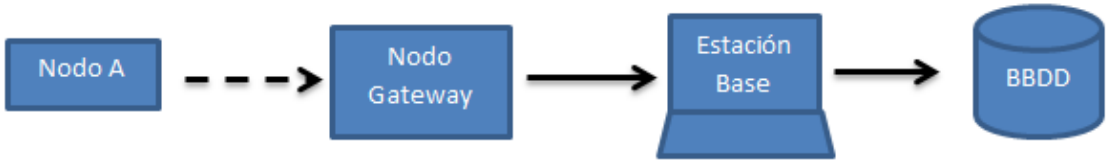

Ilustración 27: Nc6

En el Nodo A se ha instalado la aplicación envioTempTcp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

Esta es la configuración que se realiza en el Nodo A.

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::1	2001:470:1f12:e4::2	6500	6500

Tabla 65: Configuración nc6

En esta prueba se emplea el comando nc6, netcat para IPv6, el cual permite abrir puertos TCP/UDP o asociar un puerto con un intérprete Shell para forzar una conexión. El comando a emplear es nc6 2001:470:1f12:e4::1 7. Una vez introducido el comando, permite la introducción de datos por consola. Si el funcionamiento es correcto, cada vez que se escriba algo por consola obtendrá como respuesta el mismo mensaje. Es decir, si se escribe hola en la consola, debajo aparecerá un segundo hola, ya que el comando se ha establecido para realizar conexión con el puerto 7 que se encarga de devolver el mismo mensaje que llega.

Al introducir caracteres individuales se recibe el mismo carácter como respuesta, por tanto podemos concluir que funciona de manera correcta.

6.2.5.3 Navegador

El esquema a emplear será el básico:

Ilustración 28: Navegador

En el Nodo A se ha instalado la aplicación envioTempTcp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

Esta es la configuración que se realiza en el Nodo A.

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::1	2001:470:1f12:e4::2	6500	6500

Tabla 66: Configuración navegador

Para esta prueba es necesario que se abra un navegador web en la estación base con la siguiente dirección: http://[2001:470:1f12:e4::1]/read/leds.

Encendemos el sensor para que comience el flujo de mensajes entre el Nodo A y la estación base. Lo que podemos ver es cómo la página recoge el estado de los tres leds que tiene el Nodo A y nos muestra un 0 en caso de estar apagado o un 1 en caso de estar encendido.

Tras ver que esta prueba funciona podemos pasar a las dos últimas pruebas.

6.2.5.4 Comunicación entre nodos

Así, el esquema es el siguiente:

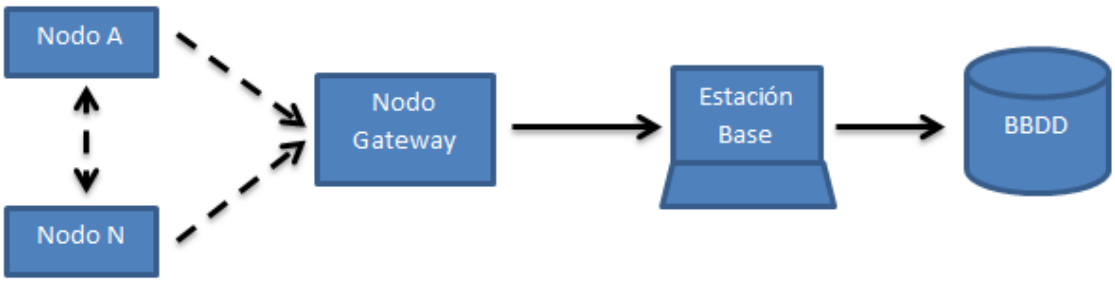

Ilustración 29: Comunicación entre nodos

El Nodo A y el Nodo B se ha instalado la aplicación envioTempTcp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

A continuación se muestra la configuración del Nodo A y del Nodo B:

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::3	2001:470:1f12:e4::2	6500	6500
B	2001:470:1f12:e4::4	2001:470:1f12:e4::3	7	6500

Tabla 67: Configuración comunicación entre nodos TCP

Al igual que ocurría en el escenario 3, tras encender los sensores se puede ver cómo la luz roja del Nodo A se enciende al recibir mensajes del Nodo B.

6.2.5.5 Comunicación fallida entre nodos

Se utiliza el siguiente esquema de red:

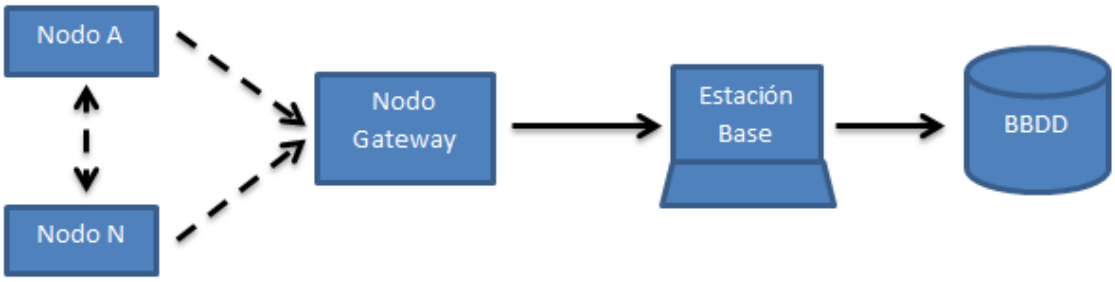

Ilustración 30: Comunicación fallida entre nodos

El Nodo A y el Nodo B se ha instalado la aplicación envioTempTcp y en el nodo Gateway se instalará IPBaseStation. En la estación base se ejecutará el fichero Listen.py para recibir los datos.

A continuación se muestra la configuración del Nodo A y del Nodo B:

Nodo	Dirección Origen	Dirección Destino	Puerto Destino	Puerto Listener
A	2001:470:1f12:e4::3	2001:470:1f12:e4::2	6500	6500
B	2001:470:1f12:e4::4	2001:470:1f12:e4::3	6500	6500

Tabla 68: Configuración comunicación entre nodos fallida TCP

En este caso, la prueba falla por no enviar el mensaje al puerto de destino correcto, como ya vimos en el escenario 3.

6.2.6 Traza con los resultados de Listener.py

A continuación se mostrará el resultado que se obtiene por pantalla al recibir los dos tipos de mensajes. En la ilustración 31 tenemos el mensaje de datos:

```
('2001:470:1f12:e4::3', 7001, 0, 0)
('Identificador nodo sensor ', 3)
('El numero de mensajes enviados por el nodo sensor es ', 9694)
('El valor obtenido por el sensor de temeratura es ', 457)
('El valor de la temperatura en grados centigrados es ', 20.662319674065088)
```

Ilustración 31: Recepción mensaje de datos

Primero, tenemos la dirección del nodo emisor. Después, los distintos campos que contiene el mensaje de datos.

Por último, el mensaje que contiene las estadísticas se puede ver en la ilustración 32:

```
('2001:470:1f12:e4::3', 7001, 0, 0)
Message <UdpReport>
  [ip.sent=0x202]
  [ip.forwarded=0x6d00]
  [ip.rx_drop=0x3]
  [ip.tx_drop=0x0]
  [ip.fw_drop=0x0]
  [ip.rx_total=0x0]
  [ip.real_drop=0x0]
  [ip.hlim_drop=0x0]
  [ip.senddone_el=0x0]
  [ip.fragpool=0x0]
  [ip.sendinfo=0x0]
  [ip.sendentry=0x0]
  [ip.sndqueue=0x0]
  [ip.encfail=0x0]
  [ip.heapfree=0x0]
  [udp.total=0x0]
  [udp.failed=0x0]
  [udp.seqno=0x0]
  [udp.sender=0x0]
  [icmp.rx=0x0]
  [route.hop_limit=0x0]
  [route.parent=0x4100]
  [route.parent_metric=0x200]
  [route.parent_etx=0xa00]
```

Ilustración 32: Recepción mensaje estadísticas

Al igual que en el mensaje de datos, lo primero que se ve es la dirección del nodo, con el identificador al final de la dirección IPv6. Después, se muestran los diferentes campos que incluye el mensaje.

6.3 Evaluación de la sobrecarga

Para calcular la sobrecarga que se produce al emplear 6LoWPAN vamos a emplear una sencilla fórmula. La sobrecarga es igual a la cantidad de información de control dividida entre la información total. Por información de control nos referimos a las distintas cabeceras que incluye el paquete y la información total es el tamaño total que tiene el paquete, incluyendo los campos de control y el payload. En 6LoWPAn los paquetes disponen de varias cabeceras, las cuales se van a enumerar a continuación.

EL paquete comienza con un tamaño máximo de 127 bytes. Primero, hemos de considerar la sobrecarga máxima de la trama, que son 25 bytes, por tanto tenemos un tamaño máximo en la capa MAC de 102 bytes. Después, hemos de considerar que se pueden incluir cabeceras de seguridad, ocupando 9, 13 o 21 bytes en función del modelo que se elija. Esto nos deja, en el peor de los casos, con 81 bytes disponibles. La cabecera IPv6 son 40 bytes, por tanto ahora solo disponemos de 41 bytes para los datos. Por último, hemos de seleccionar el protocolo de transporte, que puede ser UDP o TCP, ocupando sus cabeceras 8 y 20 bytes respectivamente.

Una vez vistos estos datos, tenemos que considerar que la sobrecarga no siempre será la misma, ya que los paquetes no siempre tendrán el tamaño máximo ni, por ejemplo, el nivel de seguridad será el mismo. Así, ahora veremos la sobrecarga en este trabajo en función de las cabeceras que usa.

En este trabajo, se ha empleado UDP, por tanto la cabecera ocupa 8 bytes. Además, la seguridad no se ha tenido en cuenta en la planificación, por tanto disponemos de 21 bytes libres adicionales. De este modo, la cantidad de información de control que tenemos es de 73 bytes. Ahora, al emplear la fórmula obtenemos un resultado de 57.48% de sobrecarga sobre el tamaño máximo del paquete.

Una vez vistos los datos podemos ver que en este caso se pierde más de la mitad del espacio del paquete para la información de control, lo cual podría mejorar si se producen avances en la compresión de los campos del paquete.

6.4 Conclusiones de las pruebas

La batería de pruebas que se ha diseñado y ejecutado a través de los distintos escenarios tiene el objetivo de establecer, en primero lugar, el grado de funcionamiento del prototipo. Además, el objetivo era también establecer las limitaciones de la implementación de 6LoWPAN a través de este prototipo.

El grado de satisfacción alcanzado a través de las pruebas ha sido muy alto, ya que todas las pruebas que trataban la interacción de los nodos sensores vía 6LoWPAN han funcionado de

manera correcta. El único escenario que no ha dado un resultado positivo es el encargado de comunicarse con un elemento que se encontraba fuera de la red, pero aun así el resultado es el esperado.

Así, podemos afirmar que 6LoWPAN funciona de manera correcta dentro de la red establecida y está limitado a su radio de acción, ya que cuando se ha intentado salir de la red no se han obtenido resultados satisfactorios. Esto se debe a que la comunicación dentro de la red se establece en las dos primeras capas, por tanto no es posible que un mensaje desde la red tenga la capacidad de llegar a un elemento que se encuentra fuera de ella.

Finalmente, comentar que tanto en UDP como en TCP se han dado resultados satisfactorios a la hora de realizar la comunicación entre los nodos sensores y la estación base, a pesar de la inestabilidad existente en algunos elementos de TCP. Por otro lado, como se ha visto con la funcionalidad web, ofrece elementos muy interesantes que pueden abrir nuevas opciones de investigación.

7. Conclusiones

A continuación se verá una revisión de los objetivos iniciales una vez que el trabajo se ha realizado. Se expondrán algunas posibilidades de futuras líneas de trabajo e investigación. Para finalizar, se incluirá el presupuesto y las conclusiones personales.

7.1 Conclusiones

En el primer capítulo, *Introducción,* se enunciaron una serie de objetivos a la hora de abordar el trabajo. Ahora, haremos un breve repaso para comprobar si se han cumplido o no y dar una valoración del trabajo realizado. Además, se mencionaran algunos posibles caminos de investigación futura que estén relacionados con este trabajo.

Primero, el empleo de 6LoWPAN como estándar de comunicación se ha, evaluado mediante un prototipo que ha funcionado a la perfección dentro de la red establecida. A pesar de que uno de los escenarios de validación del prototipo no ha sido satisfactorio, el escenario 4, no es un problema real a la hora de validar el prototipo en su conjunto ya que era una funcionalidad extra y no interfiere con el correcto funcionamiento de la comunicación dentro de la red.

El diseño de una aplicación que permita a los nodos sensores enviar los datos a la estación base también se ha cumplido, como ya hemos visto mediante la aplicación envioTemp. Se ha probado en diferentes escenarios y ha respondido a las exigencias planteadas.

El almacenamiento de los datos se ha realizado según el objetivo fijado inicialmente, mediante una base de datos. Se ha visto cómo se ha planteado y realizado el almacenamiento, por tanto este objetivo también se ha cumplido.

Por último, se establecía como objetivo un despliegue físico del prototipo, ya que es la mejor manera de validar su funcionamiento. Este punto también se ha cumplido como se ha visto, por tanto podemos concluir que todos los objetivos se han cumplido a lo largo de este trabajo.

7.2 Trabajos futuros

De cara al futuro, existen algunos elementos que pueden ser investigados para poder mejorar las funcionalidades que hemos desarrollado en este trabajo.

Primero de todo, este trabajo ha sentado una base para que a partir de aquí pueda comenzar el desarrollo de aplicaciones orientadas a redes que empleen 6LoWPAN. Este trabajo aporta, principalmente, conocimientos que servirán como punto de partida para futuras investigaciones.

La investigación de aplicaciones que empleen TCP es un inicio, aunque para que esto ocurra deben estabilizar los elementos actuales que se emplean con ese protocolo, ya que algunos

son inestables o experimentales. Para que esto ocurra puede que solo se necesite el estímulo correcto, como el control de los datos que se envían en una red de sensores vía web.

Otro posible elemento sea la inclusión de seguridad dentro del sistema, sobre todo a la hora de realizar el envío de datos desde el nodo sensor. De este modo los datos que se enviarían cifrados y no en claro, como se hace actualmente.

El control de las variables enviadas podría generar una nueva funcionalidad. De este modo, cuando el nodo sensor realiza la medición comprobará que ese dato se ajuste a unos baremos previamente seleccionados. Si, por ejemplo, al medir la temperatura el valor obtenido supera con creces las medidas habituales se podría enviar un mensaje especial que alerte de los resultados anómalos.

7.3 Presupuesto

El presupuesto estará dividido en tres secciones. En la primera se verán los costes de recursos humanos, donde se verá lo que ha costado el trabajo realizado. En la segunda, los recursos materiales, se verán los diferentes elementos que se han empleado y su coste. Por último, se verán los costes totales y se adjuntará el formulario de presupuesto.

7.3.1 Recursos humanos

Para el análisis de los recursos humanos se van a tener en cuenta las diferentes fases que se han realizado para la culminación del proyecto. Como herramienta vamos a emplear el Modelo en Cascada, a fin de poder realizar un análisis completo. Para ello, se ha dividido el trabajo en las siguientes fases: análisis del entorno del proyecto, análisis del problema, diseño, implementación, evaluación y pruebas y documentación.

El trabajo ha sido realizado por una sola persona, dedicando a la realización de este proyecto una media de 16 horas semanales. El proyecto ha sido realizado entre el 1 de abril y el 22 de septiembre. Esto hace un total de 20 semanas y alrededor de las 320 horas.

A continuación, se muestra una tabla con el coste que ha supuesto la realización de cada una de las partes considerando el número de horas empleado en cada una de ellas.

Recurso	Horas	Coste/Hora (€)	Coste fase (€)
Análisis del entorno	45	30	1350
Análisis del problema	40	30	1200
Diseño	60	30	1800
Implementación	45	30	1350
Evaluación y pruebas	80	30	2400
Documentación	50	30	1500
Total	320		9600

Tabla 69: Coste recursos humanos

7.3.2 Recursos materiales

Dentro de los recursos materiales se incluirán los elementos físicos que se han empleado para la realización de este trabajo. Los elementos necesarios han sido un ordenador de sobremesa, empleado en el laboratorio para trabajar con la red de sensores. Los diferentes componentes de la red de sensores también forman parte de esta lista. Por último, se considerará un recurso material la conexión a internet necesaria para la documentación y puesta en marcha del prototipo. A continuación se mostrarán los costes de los recursos materiales en la tabla 70:

Recurso	Cantidad	Coste (€)	Subtotal (€)
Ordenador sobremesa	1	400	400
Ordenador portátil	1	500	500
Nodo sensor MicaZ (MPR2400CA)	3	100	300
Nodo *gateway* MIB520	1	75	75
Placa de sensores MTS300CB	2	120	240
Pilas AA	4	0,75	3
Total			1518
Total amortizado			126,38

Tabla 70: Coste recursos materiales

Para calcular la amortización de los recursos empleados hemos empleado la siguiente fórmula:

$$\frac{A}{B} x\ C\ x\ D$$

Siendo:

- A el número de meses desde la fecha de facturación en que el equipo es utilizado
- B el periodo de depreciación (60 meses)
- C el coste del equipo sin IVA
- D el porcentaje de uso que se dedica al proyecto (100%)

A continuación, tenemos que hablar de los otros costes directos del proyecto, como es en este caso la conexión a internet. El precio lo encontramos en la tabla 71:

Recurso	Cantidad	Coste (€)	Subtotal (€)
Conexión a internet	5	20	100
Total			100

Tabla 71: Costes directos del proyecto

Por último, hemos de considerar que este proyecto tiene unos costes indirectos iguales al 20% del coste total del proyecto. Esto se incluirá a continuación en la tabla 72 correspondiente a los costes totales del proyecto

7.3.3 Costes totales

A continuación se muestra el coste total del proyecto, que asciende a 11792,38€, desglosado de la siguiente manera:

Recurso	Coste (€)
Recursos humanos	9600
Amortización material	126,38
Costes directos del proyecto	100
Costes indirectos	1965
Total	11792,38

Tabla 72: Costes totales

7.3.4 Plantilla del presupuesto

A continuación se muestra la plantilla en Excel empleada para la realización del presupuesto:

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Alejandro José Tébar Martín
2.- Departamento:
Informática
3.- Descripción del Proyecto:
- Titulo: **Prototipo de evaluación de una red de sensores inalámbrica basada en el protocolo 6LoWPAN**
- Duración (meses): 4
Tasa de costes Indirectos: 20%

4.- Presupuesto total del Proyecto (valores en Euros):
Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes)[1]	Coste hombre mes	Coste (Euro)	Firma de conformidad
Tébar Martín, Alejandro José		Analista	1,02	2.500,00	2.550,00	
Tébar Martín, Alejandro José		Diseñador	0,9	2.000,00	1.800,00	
Tébar Martín, Alejandro José		Programador	2,5	1.500,00	3.750,00	
Tébar Martín, Alejandro José		Documentador	2	750,00	1.500,00	
					0,00	
		Hombres mes	**6,42**	**Total**	9.600,00	

[1] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [1]
Ordenador portátil	500	100	5	60	41,67
Ordenador sobremesa	400	100	5	60	33,33
Mote MicaZ	300	100	5	60	25,00
Placa MIB520	75	100	5	60	6,25
Placa de sensores	240	100	5	60	20,00
Pilas AA	3	50	5	60	0,13
				Total	126,38

[1] Fórmula de cálculo de la Amortización:

$\frac{A}{B} x C x D$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

Ilustración 33: Plantilla presupuesto parte 1

SUBCONTRATACIÓN DE TAREAS 1518

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[1]

Descripción	Empresa	Costes imputable
Contratacion internet	Movistar ADSL	100,00
	Total	100,00

[1] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes**
Personal	9.600
Amortización	126
Subcontratación de tareas	0
Costes de funcionamiento	100
Costes Indirectos	1.965
Total	11.792

Ilustración 34: Plantilla presupuesto parte 2

7.4 Valoración personal

Valorando la parte académica de este trabajo he de reconocer que casi todos los elementos de este proyecto eran nuevos para mí. Desconocía por completo el funcionamiento de TinyOS y de nesC, que aun conociendo el lenguaje de programación C sorprende bastante su funcionamiento.

Los nodos sensores han sido un mundo totalmente nuevo, aunque muy interesante, ya que es una tecnología que puede tener un crecimiento muy amplio en ciertos sectores, y poco a poco podremos ver cómo van formando de nuestra vida si la investigación se mantiene.

Por último, puedo asegurar que ha sido un gran reto en cuanto a la investigación de las diferentes partes así como unión de todos los elementos que han formado el prototipo de red de nodos sensores. Aun así estoy contento por haber seleccionado este trabajo de fin de grado.

Bibliografía

[1] Global FutureReport, Massachusetts Institute of Technology: http://www.globalfuture.com/mit-trends2003.htm.

[1] Distributed Sensor Networks – IntroductiontotheSpecialSection: http://bit.csc.lsu.edu/~iyengar/final-papers/smc.pdf

[2] Micaz Datasheet: http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf

[3] Ad hoc On-Demand Distance Vector (AODV) Routing: http://www.ietf.org/rfc/rfc3561.txt

[4] Highly Dynamic Destination-SequencedDistance-Vector Routing (DSDV) for Mobile Computers: http://www.cs.virginia.edu/~cl7v/cs851-papers/dsdv-sigcomm94.pdf

[5] TinyOS: http://www.tinyos.net/

[6] Mote Runner: http://www.zurich.ibm.com/moterunner/

[7] Contiki-Cooja: http://www.contiki-os.org/

[8] TinyOS Programming - NesC: http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf

[9] Python: http://www.python.org/

[10] Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs): http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf

[11] Zigbee Alliance: http://www.zigbee.org/

[12] RFC IPv4 – INTERNET PROTOCOL: http://www.ietf.org/rfc/rfc791.txt

[13] IANA: http://www.iana.org/

[14] Transmission of IPv6 Packets over IEEE 802.15.4 Networks: http://tools.ietf.org/html/rfc4944

[15]Operating Rules of IEEE Project 802 WorkingGroup, CSMA/CD LANs: http://www.ieee802.org/3/rules/P802_3_rules.pdf

[16] IRIS Datasheet: http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf

Anexo A

En este anexo se verá la instalación de los elementos necesarios para poder trabajar y desarrollar el prototipo, es decir, Ubuntu y TinyOS. Primero veremos la instalación de Ubuntu en una máquina virtual y después la instalación de TinyOS versión 2.1.1.

A.1 Instalación Ubuntu

La instalación de Ubuntu se puede realizar de varias maneras. Podemos instarlo mediante un Live CD en una partición del disco duro o bien como único sistema operativo. También podemos, como se ha hecho en este trabajo, instalarlo en una máquina virtual. Esto se debe a que teniendo una imagen del sistema operativo podemos tener un entorno nuevo creado en unos minutos y, una vez termine el trabajo, podemos eliminarlo fácilmente.

Como máquina virtual hemos seleccionado VMWare, pero se puede utilizar cualquier otra. Lo primero que necesitamos es una imagen del sistema operativo. Esto se puede realizar desde la propia página de Ubuntu. En este caso, se seleccionó la versión 12.04.

Una vez lista, seleccionamos dentro de VMWare la opción “Crear una nueva máquina virtual” y marcamos la segunda opción, archivo con imagen del disco de instalación. Una vez pulsemos en siguiente, nos irá pidiendo diversos datos a lo largo de la instalación, como dónde se ha de instalar o el usuario y contraseña. Una vez pulsamos en finalizar, comenzará la creación de la máquina virtual e instantes después accederemos a ella de manera automática, donde se podrá seguir la instalación. Cuando termine nos pedirá que reiniciemos la máquina virtual. Una vez hecho, al arrancar nos pedirá que introduzcamos usuario y contraseña.

Con esto damos por finalizado la instalación del sistema operativo Ubuntu 12.04.

A.2 Instalación TinyOS

Para la instalación en Linux se puede realizar de dos maneras: mediante paquetes RPM o bien mediante la descarga desde un repositorio que contenga la versión que necesitemos. Aquí hemos optado por realizar la descarga desde un repositorio, ya que nos ha parecido más sencilla puesto que no es necesario utilizar comandos adicionales como ocurre con los paquetes RPM.

Lo primero que tenemos que hacer es seleccionar la versión que vamos a emplear. Actualmente, la última versión es la 2.1.2, ya que es la más nueva que cuenta con manuales de instalación. A pesar de esto, nosotros emplearemos la versión 2.1.1, que es la que nos permite trabajar con los nodos sensores seleccionados, ya que con la versión 2.1.2 se obtienen nuevas funcionalidades para nodos sensores con microprocesadores más potentes, como es el caso de los nodos sensores TelosB.

Lo primero que debemos hacer es seleccionar el repositorio del cual vamos a descargar TinyOS, ya que al no usar la última versión puede ser un poco complicado de encontrar. Nosotros hemos empleado el siguiente repositorio:

```
deb http://hinrg.cs.jhu.edu/tinyos karmic main
```

Una vez que tenemos el repositorio del cual vamos a realizar la descarga debemos modificar el documento `bash.bashrc`, donde se incluirá la línea del repositorio para que podamos descargarlo. Una vez hecho esto abrimos la consola para proceder actualizar el repositorio mediante `sudoapt-getupdate`. Una vez actualizado instalamos mediante:

```
sudo apt-get install tinyos-2.1.1
```

Si el repositorio que hemos seleccionado tiene todos los ficheros de TinyOS comenzará la descarga de forma normal. Una vez descargado tenemos que elegir la ubicación dentro del sistema de ficheros de Linux. Lo más usual a la hora de trabajar con TinyOS es colocarlo dentro de `/opt/` y es ahí donde vamos a colocarlo.

Además, tenemos que realizar algunas modificaciones para que las variables globales estén configuradas de forma correcta y así poder disponer de todas las funciones. Para ello accedemos al documento "tinyos.sh" e incluir las líneas que se muestran en la siguiente imagen.

```
export TOSROOT="<local-tinyos-path>"
export TOSDIR="$TOSROOT/tos"
export CLASSPATH=$CLASSPATH:$TOSROOT/support/sdk/java
export MAKERULES="$TOSROOT/support/make/Makerules"
export PYTHONPATH=$PYTHONPATH:$TOSROOT/support/sdk/python

echo "setting up TinyOS on source path $TOSROOT"
```

Ilustración 35: Variables globales

En la primera línea con "<local-tinyos-path>" nos referimos a la ubicación de la carpeta que contiene TinyOS, lo cual en este caso sería `/opt/tinyos-2.1.1`.

Después, mediante la consola introducimos el siguiente comando `sudo gedit ~/.bashrc` e incluimos la final del documento la línea `source /opt/tinyos-2.1.1/tinyos.sh`.

Ahora, una vez que tenemos las variables configuradas correctamente tenemos que pasar a configurar el resto de elementos. Podemos comprobar si están bien configuradas usando el comando `echo $TOSROOT` en la consola, lo cual debería mostrarnos la ruta `/opt/tinyos2.1.1`.

Anexo B

En este anexo se incluyen los scripts que generan la base de datos:

Bbdd.sql

```
DROP USER 'dbu';
CREATE USER dbu IDENTIFIED BY 'dbu';

DROP DATABASE ddbb;
CREATE DATABASE ddbb;
GRANT ALL ON ddbb.* TO dbu IDENTIFIED BY 'dbu';
```

Tablas.sql

```
/* Creación de la tabla datos_nodo_sensor */
USE ddbb;
DELETE FROM datos_nodo_sensor;
DROP TABLE IF EXISTS datos_nodo_sensor CASCADE;
CREATE TABLE datos_nodo_sensor (
    id_nodo_sensor int,
    modelo char(25),
    radio char(25),
    micro_controlador char(25),
    memoria_flash int,
    memoria_eeprom int,
    memoria_ram int,
    PRIMARY KEY (id_nodo_sensor)
);
/* Creación de la tabla datos_temperatura */
USE ddbb;
DELETE FROM datos_temperatura;
DROP TABLE IF EXISTS datos_temperatura CASCADE;
CREATE TABLE datos_temperatura (
    tiempo_insercion timestamp,
    id_nodo_sensor int,
    contador int,
    dato_temperatura int,
    temperaturaC float,
    PRIMARY KEY (tiempo_insercion, id_nodo_sensor)
);
```

```
/* Creación de la tabla datos_estadisticas */
USE ddbb;
DELETE FROM datos_estadisticas;
DROP TABLE IF EXISTS datos_estadisticas CASCADE;
CREATE TABLE datos_estadisticas (
    tiempo_insercion timestamp,
    id_nodo_sensor int,
    ip_sent int,
    ip_forwarded int,
    ip_rx_drop int,
    ip_tx_drop int,
    ip_fw_drop int,
    ip_rx_total int,
    ip_real_drop int,
    ip_hlim_drop int,
    ip_senddone_el int,
    ip_fragpool int,
    ip_sendinfo int,
    ip_sendentry int,
    ip_sendqueue int,
    ip_encfail int,
    ip_heapfree int,
    udp_total int,
    udp_failed int,
    udp_seqno int,
    udp_sender int,
    icmp_rx int,
    route_hop_limit int,
    route_parent int,
    route_parent_metric int,
    route_parent_etx int,
    PRIMARY KEY (tiempo_insercion,id_nodo_sensor)
);
```

Anexo C

En este anexo se incluye el contenido del programa en Python que recibe los datos y los inserta en la base de datos:

Listener.py

```
import socket
import UdpReport
import re
import sys
import MySQLdb
import math

port = 7001

if __name__ == '__main__':

    conn = MySQLdb.connect (host = "localhost", user = "dbu", passwd = "dbu", db = "ddbb")
    cursor = conn.cursor()

    s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
    s.bind(('', port))

    while True:
        data, addr = s.recvfrom(1024)

        if (len(data) > 0):

            datosSensor = list(data)

            if(ord(datosSensor[0]) == 1):

            idN = ord(datosSensor[1]) << 8 | ord(datosSensor[2])
            contador = ord(datosSensor[3]) << 8 | ord(datosSensor[4])
            dato = ord(datosSensor[5]) << 8 | ord(datosSensor[6])

            a = 0.00130705
            b = 0.000214381
            c = 0.000000093
            r1 = 10000
            t = 0
            adc_fs = 1023
            tempReal = 0

            adc = dato
            Rthr = r1 * (adc_fs - adc) / adc
            Rthr1 = math.log(Rthr)
            Rthr3 = math.pow(Rthr1,3)
            b=b*Rthr1
            c=c*Rthr3
            t=a+b+c
            tempK = 1/t
            tempC = tempK - 273
```

```
print ("Identificador nodo sensor ", idN)
print ("El numero de mensajes enviados por el nodo sensor es ", contador)
print ("El valor obtenido por el sensor de temeratura es ", dato)
print ("El valor de la temperatura en grados centigrados es ", tempC)

cursor.execute('''INSERT INTO datos_sensor (id_nodo_sensor,contador,dato_temperatura,
temperaturaC) values (%s,%s,%s,%s)''',(idN,contador,dato, tempC))

if(ord(datosSensor[0]) == 2):

rpt = UdpReport.UdpReport(data=data, data_length=len(data))
    print rpt
    print addr

ipSent = ord(datosSensor[1])
ipForwarded = ord(datosSensor[2])
idN = ord(datosSensor[3]) << 8 | ord(datosSensor[4])
ipRxD = ord(datosSensor[5])
ipTxD = ord(datosSensor[6])
ipFwD = ord(datosSensor[7])
ipRxT = ord(datosSensor[8])
ipRealD = ord(datosSensor[9])
ipHlimD = ord(datosSensor[10])
ipSendDone = ord(datosSensor[11])
ipFragPool = ord(datosSensor[12])
ipSendInfo = ord(datosSensor[13])
ipSendEntry = ord(datosSensor[14])
ipSendQueue = ord(datosSensor[15])
ipEncfail = ord(datosSensor[16])
ipHeapFree = ord(datosSensor[17]) << 8 | ord(datosSensor[18])
udpTotal = ord(datosSensor[19]) << 8 | ord(datosSensor[20])
udpFailed = ord(datosSensor[21]) << 8 | ord(datosSensor[22])
udpSeqno = ord(datosSensor[23]) << 8 | ord(datosSensor[24])
udpSender = ord(datosSensor[25]) << 8 | ord(datosSensor[26])
icmpRx = ord(datosSensor[27]) << 8 | ord(datosSensor[28])
routeHopLimit = ord(datosSensor[29])
routeParent = ord(datosSensor[30]) << 8 | ord(datosSensor[31])
routeParentMetric = ord(datosSensor[32]) << 8 | ord(datosSensor[33])
routeParentEtx = ord(datosSensor[34]) << 8 | ord(datosSensor[35])

cursor.execute('''INSERT INTO datos_estadisticas (idNodo, ip_sent, ip_forwarded,
ip_rx_drop,ip_tx_drop, ip_fw_drop, ip_rx_total, ip_real_drop, ip_hlim_drop, ip_sendDone_el,
ip_fragpool, ip_sendinfo, ip_sendentry, ip_sendqueue, ip_encfail, ip_heapfree, udp_total,
udp_failed, udp_seqno, udp_sender, icmp_rx, route_hop_limit, route_parent,
route_parent_metric, route_parent_etx) values
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)''',(idN, ipSent,
ipForwarded, ipRxD, ipTxD, ipFwD, ipRxT, ipRealD, ipHlimD, ipSendDone, ipFragPool, ipSendInfo,
ipSendEntry, ipSendQueue, ipEncfail, ipHeapFree, udpTotal, udpFailed,udpSeqno,udpSender, icmpRx
, routeHopLimit, routeParent, routeParentMetric, routeParentEtx))
```

Anexo D

En este anexo se encontrará el código de la aplicación instalada en los nodos sensores:

Makefile

```
SENSORBOARD=mts300
COMPONENT=envioTempC

# uncomment this for network programming support
# BOOTLOADER=tosboot

CFLAGS += -DCC2420_DEF_CHANNEL=15
# CFLAGS += -DRF230_DEF_CHANNEL=15
# CFLAGS += -DCC2420_DEF_RFPOWER=4

# disables support for the AM stack, which somewhat reduces code size
# and compresses packet formats.  If you want to use other tinyos
# protocols which are AM-based, you should not include this.
# CFLAGS += -DIEEE154FRAMES_ENABLED


# lib6lowpan contains inet_ntop6 and inet_pton6 to process ascii
# representations of IPv6 addresses.  You can remove them to save some
# code if you don't use them
# CFLAGS += -DNO_LIB6LOWPAN_ASCII

# if this is set, motes will send debugging information to the address
# listed.
 CFLAGS += -DREPORT_DEST=\"2001:470:1f12:e4::2\"


# printf debugs.  works only on telosb/tmote sky
# CFLAGS += -DPRINTFUART_ENABLED


include $(MAKERULES)
```

UDPReport.h

```
#ifndef _UDPREPORT_H
#define _UDPREPORT_H

#include "Statistics.h"

nx_struct udp_report {
  nx_uint8_t tipoMensaje;
  nx_uint16_t seqno;
  nx_uint16_t sender;
  ip_statistics_t    ip;
  udp_statistics_t   udp;
  icmp_statistics_t  icmp;
  route_statistics_t route;
};

nx_struct data_msg {
  nx_uint8_t tipoMensaje;
  nx_uint16_t sender;
  nx_uint16_t seqno;
  nx_uint16_t datos;
} ;

#endif
```

envioTempC.nc

```
#include <6lowpan.h>

configuration UDPEchoC {

} implementation {
  components MainC, LedsC;
  components envioTempP;

  envioTempP.Boot -> MainC;
  envioTempP.Leds -> LedsC;

  components new TimerMilliC() as Data;
  components new TimerMilliC() as Statistics;
  components IPDispatchC;

  envioTempP.RadioControl -> IPDispatchC;
  components new UdpSocketC() as Status,
    new UdpSocketC() as Echo;
  envioTempP.Echo -> Echo;

  envioTempP.Status -> Status;

  components new TempC() as Sensor;
  envioTempP.Read -> Sensor;

  envioTempP.StatusTimerS -> Statistics;
  envioTempP.StatusTimerD -> Data;

  components UdpC;
  envioTempP.IPStats -> IPDispatchC.IPStats;
  envioTempP.UDPStats -> UdpC;
  envioTempP.RouteStats -> IPDispatchC.RouteStats;
  envioTempP.ICMPStats -> IPDispatchC.ICMPStats;

  components UDPShellC;
}
```

envioTempP.nc

```
#include <lib6lowpan.h>
#include <ip.h>
#include <IPDispatch.h>

#include "UDPReport.h"
#include "PrintfUART.h"

#define REPORT_PERIOD_S 300
#define REPORT_PERIOD_D 100

module envioTempP {
  uses {
    interface Boot;
    interface SplitControl as RadioControl;

    interface UDP as Echo;
    interface UDP as Status;

    interface Leds;
    interface Read<uint16_t> as Read;

    interface Timer<TMilli> as StatusTimerS;
    interface Timer<TMilli> as StatusTimerD;

    interface Statistics<ip_statistics_t> as IPStats;
    interface Statistics<udp_statistics_t> as UDPStats;
    interface Statistics<route_statistics_t> as RouteStats;
    interface Statistics<icmp_statistics_t> as ICMPStats;
  }

} implementation {

  bool timerStartedD;
  bool timerStartedS;

  nx_struct udp_report stats;
  nx_struct data_msg datos;

  struct sockaddr_in6 route_dest;

  event void Boot.booted() {
    call RadioControl.start();
    timerStartedD = FALSE;
    timerStartedS = FALSE;
```

```
#ifdef REPORT_DEST
    route_dest.sin6_port = hton16(7001);
    inet_pton6(REPORT_DEST, &route_dest.sin6_addr);
 #endif

    call StatusTimerS.startPeriodic(REPORT_PERIOD_S);
    call StatusTimerD.startPeriodic(REPORT_PERIOD_D);

    dbg("Boot", "booted: %i\n", TOS_NODE_ID);
    call Echo.bind(7);
    call Status.bind(7002);
  }

  event void RadioControl.startDone(error_t e) {

  }

  event void RadioControl.stopDone(error_t e) {

  }

  event void Status.recvfrom(struct sockaddr_in6 *from, void *data,
                             uint16_t len, struct ip_metadata *meta) {

  }

    event void Echo.recvfrom(struct sockaddr_in6 *from, void *data,
                             uint16_t len, struct ip_metadata *meta) {

    call Echo.sendto(from, data, len);
  }

  event void Read.readDone(error_t result, uint16_t data) {
    if (result == SUCCESS){
      datos.datos = data;
      call Leds.led0Toggle();
    }
   }
```

```
  event void StatusTimerD.fired() {
    if (!timerStartedD) {
    call StatusTimerD.startPeriodic(REPORT_PERIOD_D);
      timerStartedD = TRUE;
      }

    call Read.read();

    datos.tipoMensaje=1;
    datos.seqno++;
    datos.sender = TOS_NODE_ID;

    call Status.sendto(&route_dest, &datos, sizeof(datos));

    call Leds.led2Toggle();
timerStartedD=FALSE;
  }

  event void StatusTimerS.fired() {
   if (!timerStartedS) {
    call StatusTimerS.startPeriodic(REPORT_PERIOD_S);
      timerStartedS = TRUE;
      }

    stats.tipoMensaje=2;
    stats.seqno++;
    stats.sender = TOS_NODE_ID;

    call IPStats.get(&stats.ip);
    call UDPStats.get(&stats.udp);
    call ICMPStats.get(&stats.icmp);
    call RouteStats.get(&stats.route);

    call Status.sendto(&route_dest, &stats, sizeof(stats));

    call Leds.led1Toggle();
    timerStartedS=FALSE;
  }
}
```